D2P: From Recursive Formulations to Distributed-Memory Codes

Nikhil Hegde*

Computer Science and Engineering
Indian Institute of Technology
Dharwad
Karnataka, India
nikhilh@iitdh.ac.in

Qifan Chang

Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
qchang@purdue.edu

Milind Kulkarni

Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
milind@purdue.edu

ABSTRACT

Recursive formulations of programs are straightforward to reason about and write, often have good locality properties, and readily expose parallelism. We observe that it is easier to automatically generate distributed-memory codes for recursive formulations with certain properties: i) inclusive—a recursive method’s parameters summarize the data access done within the method body. ii) Intersection—data-set intersection tests among method invocations can be computed efficiently. In this paper we present D2P, a system that automatically generates distributed-memory codes for recursive divide-conquer algorithms with these properties. D2P produces MPI-based implementations starting from shared-memory specifications of the recursive algorithms. We evaluate D2P with recursive Dynamic Programming (DP) algorithms, since these algorithms have the desired properties and are well known. We show that the generated implementations are scalable and efficient: D2P-generated implementations execute faster than implementations generated by recent distributed DP frameworks, and are competitive with (and often faster than) hand-written implementations.

CCS CONCEPTS

• **Software and its engineering** → **Distributed programming languages; Recursion**; • **Theory of computation** → *Dynamic programming*.

KEYWORDS

Recursion, Distributed memory, Inspector-executor

ACM Reference Format:

Nikhil Hegde, Qifan Chang, and Milind Kulkarni. 2019. D2P: From Recursive Formulations to Distributed-Memory Codes. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19)*, November 17–22, 2019, Denver, CO, USA. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3295500.3356205>

*This work was performed while Nikhil Hegde was in the School of Electrical and Computer Engineering at Purdue University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SC '19, November 17–22, 2019, Denver, CO, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-6229-0/19/11...\$15.00

<https://doi.org/10.1145/3295500.3356205>

1 INTRODUCTION

Programmers like recursive formulations of programs because they are straightforward to reason about and write, often have good locality properties, and readily expose parallelism [10, 27]. We want to create distributed parallel implementations *automatically* from simple shared-memory recursive specifications. We observe that the automation process is simplified when recursive formulations have certain properties: i) the data-dependencies of a recursive method are inclusive (i.e. the collective data-dependencies of all the recursive sub-invocations within the body of the method are a subset of the dependencies captured by method’s arguments). ii) The data set intersection tests necessary for determining inter-task dependencies are simplified.

Creating distributed-memory implementations involves several challenges that do not arise in shared-memory systems. First, we need to partition the data and computation into *tasks* of appropriate granularity split among different compute nodes so that different nodes can process portions of the data in parallel. Second, we need to insert communication between nodes to satisfy data dependencies between tasks. Finally, we need to schedule tasks efficiently, to balance parallelism with the communication overhead. Automating the creation of distributed-memory programs requires that these challenges are automatically handled.

In recursive formulations with the inclusive property, each recursive “task” (think: invocation) touches a subset of the data of its parent task (think: caller method). Hence, coarse-grained tasks, suitable for a task-parallel model of computation, can be easily identified. When the tasks follow a specific order in computing the data sets, which either fully overlap or are disjoint, the data set intersection tests are greatly simplified. Identifying that these properties in recursive divide-conquer formulations can simplify distributed-memory code generation is the main novelty of our work.

A specific set of programs that are important and satisfy these properties are recursive Dynamic Programming (DP) [5] programs. DP algorithms are very efficient in solving problems arising in domains such as bio-informatics, mobile communication, and finance. As some of these problems involve large data sizes that can exceed the memory capacity of a single compute node, distributed-memory solutions become necessary to process the entire data. Because formulating an efficient algorithm is a different challenge than performing data partitioning and task creation, communication insertion, and task scheduling, we would like a system where the concerns of *creating* an algorithm can be separated from the concerns of *distributing* it. In particular, we would like a system that automates as much of the distribution process as possible, allowing programmers to focus on simply designing algorithms.

Figure 1: D2P system overview.

While much of prior work has addressed the challenges of automating the distribution in different contexts for iterative codes of general programs, recursive programs are not explored as extensively. A majority of existing systems [3, 6, 32, 33, 40] work on iterative codes. In case of recursive programs, they adopt a profiling-based approach and assume a regular communication pattern [33]. Programmer created annotations in recent research exploits the inclusive property to optimize for locality and parallelism of recursive divide-conquer programs on shared-memory [27] and heterogeneous systems [4]. In case of DP programs, a majority use iterative codes. While most parallel implementations are for shared-memory systems, there exist only a few hand-tuned distributed-memory implementations [1, 2, 21, 28, 39, 41, 42]. Recent work has shown that efficient recursive formulations of DP algorithms outperform iterative codes on shared-memory systems primarily because of better cache utilization [11, 12]. Regarding distributing the computation, recursive DP algorithms involve irregular communication patterns and offer a specialized (or simplified) domain. Hence, previous techniques for automation are not effective. Importantly, the inclusive and intersection properties are *implicit* in these recursive DP formulations and this is the key insight we exploit in *automatically* creating distributed-memory implementations of recursive DP algorithms.

1.1 Overview

In this paper, we present D2P, a framework that takes a shared-memory specification of a recursive algorithm with the inclusive and intersection properties (without any parallelism or partitioning specifications), and produces an MPI based, distributed-memory parallel code with recursive methods.

Figure 1 shows the overview. The input to D2P code generator is a specification, which is a high-level outline of a recursive algorithm. The specification need not mention the recursion base cases, which are application-specific. This attribute of the specification points to an important way in which D2P differs from other fully-automated code generation schemes [3, 4, 26, 32, 33]: user-assistance is necessary in filling the D2P-produced implementation with application-specific details to make it complete. Despite this semi-automatic nature, D2P completely automates (and hides) the challenging aspects of distributed-memory programming and delegates only the algorithm design aspects to an application designer.

At compile time, the D2P Codegen generates codes for the inspector and executor modules from the specification. Every process in a multi-process execution of the program executes these

modules. The inspector and executor codes are parametrized on a data-partitioning (mapping) scheme, input size, and the number of processes, to enable the creation of an application code providing portable-performance. These parameters are provided at runtime. The inspector contains code for figuring out process-specific computations (method invocation with specific argument values) and the executor code contains rules (method bodies with communication directives) for performing those computations.

At runtime, the inspector module of every process determines tasks to compute, identifies task dependencies, establishes communication channels for each task, and prepares a schedule based on the identified task dependencies. The executor module of a process then executes the schedule.

First, the inspector *unrolls* a top-level recursive method multiple levels to create tasks. The unrolling hierarchically decomposes (breaks) the data into smaller parts and implicitly creates a tree. A vertex of the tree represents recursive method invocation computing a part resulting from the repeated decomposition up to the vertex’s level. The children of this vertex are again method invocations called from within the body of the vertex’s method. The internal vertices of the tree are *not involved* in computing the data: they only decompose the data into finer parts and in some cases do minimal work, such as initializing, before any of the children modify the data (due to the intersection property). Therefore, all the computation is done by leaf method invocations and hence, the leaves of the tree represent the tasks in D2P.

Based on a mapping scheme, the tasks are partitioned among processes in a deterministic manner known to all processes (facilitating determination of communication later). Task ownership follows data ownership as the processes follow an owner-computes rule (single owner for a data region) during computation. As method arguments are the only (entire) data read and written within the invocation (due to the inclusive property mentioned above), it is sufficient if the leaf method arguments are inspected to determine inter-task dependencies. Unique IDs associated with data regions simplify this dependency computation. Finally, the inspector uses dependence information to insert communication—filling holes in communication directives, and identifying and establishing communication channels between processes to send/receive data as demanded by the task allocation.

At the end of inspector’s execution, every process has a list of tasks. The executor begins executing a task from the list for which all the dependences are satisfied. As soon as a task completes execution, the executor sends the results of the computation to processes that own dependent tasks. Dependent task owners go through their respective task list to update task dependences and execute a task if the task dependences are fully satisfied. The result is a “single program multiple data” (SPMD) implementation employing asynchronous communication capable of running on a distributed-memory system.

The contributions of this paper are:

- We provide a framework D2P, which can take a specification of any recursive divide-conquer algorithm with the inclusive and intersection properties and convert the specification to a distributed-memory implementation capable of exploiting node- and core-level parallelism.

- We evaluate our framework on a set of recursive formulations having these properties: recursive DP algorithms. As an added feature, we augment D2P’s code generator to parse specifications produced by an existing tool [10] that can automatically generate specifications for a *subset* of the recursive DP algorithms. Thereby, translating specifications of a wide class of DP problems to their respective MPI+Cilk-based implementations.
- We evaluate the scalability of D2P-generated implementations, preprocessing overheads, and compare against existing distributed-memory implementations.

Our evaluation shows that D2P implementations scale well and can admit extremely large problem sizes much beyond the memory capacity of a single node. We also observe that the overheads of task partitioning (task creation, assignment, and dependency determination) are negligible compared to the actual computation. Finally, we find that a D2P-generated program scales better than other distributed-memory implementations: the D2P-generated implementation of Smith-Waterman runs 59× to 412× faster than an implementation generated by an existing framework for parallelizing *iterative* DP formulations on distributed-memory systems. Also, the D2P-generated implementation outperforms a *hand-written*, application-specific implementation of Smith-Waterman in *most cases*, and is only 27% slower in the worst case.

2 BACKGROUND AND MOTIVATION

Recursive algorithms with the inclusive and intersection properties, in particular, have advantages when it comes to distributed-memory implementations: every recursive method invocation accesses a data set, which is the super-set of the union of the data sets accessed within all recursive invocations done in the method body. In other words, the method arguments specify a bound on the entire data regions read/written, thus avoiding the need for programmer specified annotations. This also makes it easier to reason about data-dependencies necessary for automating parallelism extraction and data-partitioning. In comparison, dependency analysis is complicated in *iterative* codes with multiple layers of `for` loops with regular and irregular accesses. A recursive method invocation represents a coarse-grained computation and hence, a suitable target for the creation of a task. The recursive nature also exposes intra-task parallelism, which is naturally adapted by a task-parallel model of computation such as the Cilk [18] system. In recursive algorithms with the intersection property, a hierarchical decomposition (due to recursive calls) creates disjoint (unique) data partitions. Furthermore, the partitions are input-independent, computed as per a preorder execution of the recursion tree vertices, and can be represented by unique identifiers. As a result, inter-task dependences are computed easily.

Efficient recursive formulations—hereafter referred to as recursive formulations—of DP algorithms have additional advantages: i) control partitioning is not a concern due to the absence of branches in the formulation, ii) a static task partitioning scheme (due to a predictable amount of computation per task) can simplify communication insertion, and iii) simple, and unique identifiers (Z-order [29]

numbers) can be assigned to data partitions resulting from hierarchical decomposition of the DP table. We next describe an example DP algorithm and its recursive DP formulation.

Minimum weight triangulation (MWT) [25] is commonly used in applications in finite element method and computational geometry, among others, to triangulate a convex polygon. The goal in MWT is to partition the polygon into triangles such that the edges do not intersect and the sum of the edge lengths of the component triangles is minimized. A dynamic programming (DP) based approach to solving the MWT problem divides the problem into subproblems, computes solutions to the subproblems, and combines the optimal solutions of smaller subproblems to find an optimal solution for the whole problem. The recurrence equation in Figure 2(a) systematically computes the solution for the MWT problem. The intuition behind this equation is to partition the polygon into a simple triangle ($W(i, j, k)$) and sub-polygon(s) to the left and/or right ($C(i, k)$, $C(k, j)$) of the simple triangle, compute the solutions for the sub-polygon(s) *separately*, then combine the solutions (*overlapping subproblems*). If the sub-polygons have an optimal solution, then the combined solution is *guaranteed* to be optimal (*optimal substructure*). The optimal substructure and overlapping subproblems properties are fundamental to problems that admit DP solutions [14].

The standard approach to implementing the recurrence equation is to use an “iterative” formulation. This consists of a set of nested `for` loops that iteratively compute the cells of a two-dimensional matrix (DP table) storing the solutions of all the subproblems computed. Figure 2(a) also shows the iterative code and its operation.

Every iteration of the `for` loop on line 4 can be run in parallel in this code. This parallelizing scheme reflects *wavefront-parallelism*, which is a popular approach for parallelizing iterative codes. The iterative code shown performs a bottom-up computation of finding optimal solutions to all subproblems of smaller sizes first before moving on to a bigger subproblem (computing all cells on the main diagonal before computing cells on the smaller adjacent diagonal). This computation order results in unpredictable runtimes, especially in the presence of cache sharing [10], and exhibits poor temporal locality compared to a recursive formulation of the MWT problem.

3 DESIGN

D2P uses a specification of a recursive computation to generate a distributed implementation. We first describe what this specification looks like (Section 3.1). We then describe how D2P uses an inspector/executor approach to build a distributed implementation (Section 3.2). The inspector and executor are executed on each process in a SPMD manner. Finally, we describe some design details for how the inspector and executor work (Section 3.3). Examples of the spec provided to D2P and the code generated by D2P are provided as extension to AD/AE appendices.

3.1 Specification

The input to D2P is a specification, which is a high-level outline of a recursive algorithm. Figure 2(b) shows a specification, which is a recursive formulation for the MWT problem. The recursive formulation hierarchically breaks the DP table into smaller quadrants and computes their solutions. A D2P specification consists of a set of recursive methods, among which there is only one top-level

Figure 2: The Minimum Weight Triangulation problem: (a) shows the recurrence equation to compute the least cost of triangulating a convex polygon. The Figure also shows the standard implementation scheme using an iterative code. (b) shows an example specification in D2P. (c) shows the operation of the recursive formulation outlined by the specification.

recursive method that calls other methods. We assume that the top-level recursive method is always specified first, followed by other methods. A method's definition starts with the method signature followed by the method body. Empty lines separate method definitions. Further, we assume that the first parameter of a method represents the data computed (written) and the remaining parameters represent the data read in order to compute the data. We also assume that the argument names in a recursive method call identify the data regions accessed (e.g. $X00$ in $A(<X00>)$ in the second line of Figure 2(b) identifies the *top-left* quadrant of a two-dimensional matrix X). The recursion base cases of the methods are omitted from the specification for the sake of simplicity.

The execution begins with a call to the top-level recursive method A , with the argument X representing the entire DP table. The call to method A implicitly breaks the problem into smaller subproblems (2×2 tiles of an example 4×4 matrix), represented as quadrants ($X00$ - $X11$). The recursive calls to methods A , B , and A in the body of method A compute the first, second, and fourth quadrants ($X00$, $X01$, and $X11$) respectively. Note that the recurrence equation in Figure 2(a) specifies that only the upper triangular matrix is computed. Hence, only these three quadrants are computed. We expect the binary number used in the identifier names (X_{yy} , where yy is a two-digit binary number) to identify the quadrant and the DP table dimension (2D in MWT) as well.

Each of these recursive calls further breaks the quadrants into smaller parts (1×1 tiles identified by U_{yy} , V_{yy} , and Z_{yy}). For example, the call to $B(X01, X00, X11)$ in method A 's body (line 4) computes the second quadrant $X01$ by reading quadrants $X00$ and $X10$. To compute $X01$, method B first breaks $X01$ (and $X00$, $X10$ as well) into smaller quadrants. The first call within method B 's body, $B(Z10, U11, V00)$, illustrates this: it computes the third quadrant $Z10$ of Z , which is the same as $X01$, by reading the fourth quadrant $U11$ of U , and the first quadrant $V00$ of V (U and V are the same as $X00$ and $X11$ resp.). The recursion stops here because it is not possible to decompose the 1×1 tiles (cell) further.

The base case defines how a cell or a tile (group of cells) is computed from other cells (lines [8-10] in `COST` method of Figure 2(a)). Method signatures of B and C in the specification tell that these methods compute a cell (tile) by reading two cells (tiles), U and V . Hence, the base case is same for both these methods. Method A computes a cell from initial conditions ($W(i, i+1, i+2)$) and hence, has no dependencies. Therefore, A 's base case is different from methods B and C . By default, the base case computes a single cell in D2P. Note that in addition to the base case, the specification omits: i) input size necessary to identify the exact data region within the 2D matrix (e.g. a 2×2 square tile containing cells from $(0, 2)$ to $(1, 3)$ — $X01$ and ii) the data-type of the cell (`CELLTYPE`—`float`/`double` for the MWT problem). The end-user is expected to insert code corresponding to these in the D2P generated code (see Section 4, auxiliary methods, for details.). As the specification does not contain any DP-specific properties, D2P could generate code for any recursive divide-conquer algorithm with the inclusive and intersection properties. Next, we explain how D2P converts a specification to a distributed-memory parallel code.

3.2 Inspector and Executor

A task in D2P computes a method call with specific argument values. Therefore, a task constitutes: i) a pointer to the method, ii) argument values (iii) input, and (iv) output dependences. The input dependences are a set of task IDs and the output dependences are a set of process IDs. The job of determining what the tasks should be and what the dependences are is delegated to an inspector that D2P generates. Executing the tasks while respecting these dependences is delegated to an executor that D2P generates. We next describe at a high level how these inspectors and executors are generated for each process. Details about how the inspector determines which tasks a process owns, and what the task's dependences are, are described in Section 3.3.

The goal of the inspector, when executed, is to produce a list of tasks, where each task computes a data region based on reading other regions, as shown in Figure 3. The inspector does so by

insert code to:

```
I. Generate a map M of data region IDs to task IDs: data region ID
is the ID of the write parameter, and the task ID is that of
the last task that wrote the region.

II. for each method m A(X, Y, Z) defined in spec:
1. Generate inspector method signature
  a. Produce full types for each parameter.
  b. Each parameter also has metadata parameters of bounding-
  box and ID of the data region that a parameter represents.
  c. Add stack depth parameter to track unroll depth.
2. Generate inspector base case
  a. Base case triggers when stack depth = unroll depth.
  b. Generate unique task ID.
  c. Insert region ID (X's ID) and task ID into map M.
  d. Based on X's ID, decide which process owns a task writing to X
  i) If self (i.e. process Pself)
    - Create a task T.
    - Figure out input and output dependences for task T.
    input deps:
    - Use map M to find TlastID, ID of the last task that wrote X.
    - Add TlastID to T's input deps.
    - Use the list of tasks to get Tlast, the task with ID TlastID.
    - Add T's owner process ID (i.e. Pself) to Tlast's output deps.
    /* tasks are created in execution order. So, this creates
    correct input dependences. For the same reason, T's output
    dependences are not known yet. They will be figured as
    explained in the next step */
    - Add T to Pself's list of tasks
  e. for each read parameter R: {Y, Z}
  i) If R is not the same as write parameter X:
    - Use map M to find TlastRID, ID of the last task that wrote R.
    - Process Pself adds TlastRID to T's input deps
    - Based on R's ID, decide which process owns TlastR,
    the last task that wrote R. Let PownsR be the owner.
    - Process PownsR adds Pself to TlastR's output deps.
3. Generate recursive case
  a. Create sub regions (e.g. X00, Y01, Z11), their bounding-boxes,
  and compute their region IDs.
  b. Call inspector methods as per the spec (use arguments created
  in step a).

III. Generate wrapper that calls the top-level inspector method:
1. Create arguments to call the top-level inspector method.
  - Bounding-box for the data region (whole matrix): this depends
  on input size (size). For a 2-dimensional grid, cells(0,0) to
  (size-1, size-1) are computed.
  - data region: nullptr
  - data region ID: 0
  - call stack depth: 0
  /* Since the computation is distributed among tasks, memory is
  not allocated to the whole matrix (data region is nullptr).
  As tasks compute sub-data regions, memory for a sub-data region
  is allocated when the first task that writes to that
  sub-data region is created */
2. Call top-level inspector method with arguments created.
```

Figure 3: Operation of the D2P inspector generator

constructing a set of mutually-recursive functions, corresponding to the functions in the specification. These functions execute at runtime to generate the list of tasks. When the recursion reaches a leaf, this corresponds to a task that must be executed. If this task is owned by the current process, the inspector computes the dependences for this task and places it in the process's task list (Section 3.3 describes how task ownership is determined and how dependences are computed). When the inspector module of each process completes execution, all the task dependencies are computed and each process gets a per-process list of tasks. Note that

insert code to:

```
I. for each method m A(X, Y, Z) defined in spec:
  1. Same as II.1 in Figure 3 except each parameter has one
  metadata parameter of bounding-box.
  2. Same as II.2 in Figure 3 except base case triggers when
  bounding-box size=1. Also the base case is empty and needs to be
  filled-in by the end-user.
  3. Same as II.3 in Figure 3 except generate two versions of the
  code(II.3.b) corresponding to serial and parallel execution.
  a. parallel version triggers when call stack depth < SPANNCUTOFF.
  - Insert cilk_spawn and cilk_sync appropriately for calls that
  can be executed in parallel.
  /* parallelism is specified through annotations now */
  b. Serial version triggers when call stack depth >= SPANNCUTOFF.

II. Generate wrapper that listens for incoming messages:
  1. Scan the task list for a task with all dependences satisfied.
  2. if there exists such a task T
    a. Execute task T.
    b. When done, write T's ID in the metadata of the data
    region that T computed. Send data region + metadata to
    processes mentioned in T's output deps.
    c. Remove T from task list
  3. If the task list is empty stop listening.
  3. Otherwise, block on (listen to) incoming messages
  - Triggers when there is an incoming message
  a. Retrieve producer task ID Tprd from metadata in the message
  b. For each task T in list:
    - If Tprd is in T's input deps:
      - Remove Tprd from input deps.
      - Set T's argument value to message received.
  goto step II.1
```

Figure 4: Operation of the D2P executor generator

because the inspector generates tasks only from leaves of this recursive structure, it is essentially *unrolling* the recursion of the specification to identify leaf tasks.

The goal of the executor is to execute the list of tasks that the inspector produced. Figure 4 shows how the skeleton for that executor is constructed. The executor module of each process spins in a loop scanning the task list produced by the inspector. A task is executed when either all its input dependences are satisfied or the set of input dependences is empty. When a task completes execution, it is removed from the list and the executor immediately sends the results of the computation (a data region / tile in a matrix) to all the processes specified in the output dependences. Before sending the results, the executor attaches additional metadata that indicates the task ID that computed the results and the bounding box information of the tile computed. The executor waits (blocks on a wildcard message) when there are no tasks to execute.

Upon receiving a message, the executor updates the input dependencies for all tasks waiting on the task mentioned in the metadata received. The executor then executes any task for which all the dependences are satisfied. The executor exits the loop when the list of tasks is empty.

3.3 Design Details

We next explain inspector's operation at runtime. Figure 5 illustrates task creation, dependency inference, and task partitioning.

Unrolling depth. Recall that the inspector unrolls the computation to a certain depth until it arrives at base cases that it turns into tasks for the executor. With a level of unrolling, the DP table is broken into smaller parts, so more unrolling leads to smaller, but more

Figure 5: D2P inspector in action: (a) shows task creation by unrolling MWT’s top-level recursive method *A* two levels deep. The leaves of the recursion tree are identified as tasks. The Figure also shows the numbers of tiles read and written by each task (numbering follows from Figure 2(a)). (b) shows inter-task dependences. (c) shows task partitioning among 4 processes.

numerous tasks. Deciding the optimal granularity of these tasks—i.e., number of levels of recursion to unroll—is hard. Typically, when executing a recursive parallel program on a single compute node, the tradeoff is between the overhead of creating new tasks and the amount of parallelism exposed. For a distributed-memory implementation, the number of levels of recursion to unroll, D , reflects a tradeoff between communication overhead and the available parallelism: a larger D introduces more tasks, which may increase the available parallelism but also introduces more data dependences, and hence communication. Figure 5(a) shows the tasks created when $D=2$.

Identifying data dependences via effect intersection. Figure 5(b) shows the data dependences among tasks resulting from unrolling. Correct dependences can be derived after computing a linearization of the tree in preorder traversal (to ensure that dependences are resolved in program order). The dependency structure, as the arrows indicate, can be computed using simple set intersection (of the tile numbers) given the tile numbering information. The source of the arrow is a task writing to a DP table tile (producer) and the destination is a task that reads these tiles (consumer). Enforcing these data dependences is all that is necessary to ensure correct execution, due to the inclusion property of the dependences as described earlier.

Note, interestingly, that because D2P performs unrolling of the original recursive algorithm, it is able to expose *more* parallelism than the original formulation. This is because the original formulation uses control dependences (in the form of sequential ordering) to conservatively enforce data dependences that arise between coarse-grained tasks. Two coarse-grained tasks x and y may have to execute sequentially because some subtask in y is dependent on some subtask in x , even though *other* subtasks in y can execute in parallel with x . As a concrete example, note that in the original recursive formulation of MWT in Figure 2(c), the first *B* task executes sequentially after the parallel execution of the first two *A* tasks. However, in the unrolled dependence graph of Figure 5(b), we see that this is conservative: t_7 (arising from a *B* task) can execute in parallel with t_3 and t_6 (arising from *A* tasks) despite the control dependence in the original formulation. D2P enforces exactly these

data dependences, and hence is able to expose more parallelism than the original parallel recursive implementation, depending on how much unrolling is done. Of course, performing additional unrolling can incur additional runtime overhead and communication, so this tradeoff must be carefully managed.

Task assignment. D2P uses an *owner-computes* strategy: the inspector assigns DP table tiles to processes as per the mapping strategy provided at runtime, and then *all* tasks that compute the tiles mapped to a process are assigned to that process. This ensures that the task-to-process assignment results in a single owner for a tile. Note that this simplifies communication, as we do not need to consider different processes writing to the same tile. D2P provides knobs to programmers to control the mapping strategy (see Section 4). Figure 5(c) shows an example mapping strategy of assigning tiles along the diagonals to four processes in round-robin order. The Figure also shows the resulting tile and task distributions.

Communication insertion and data movement. In a distributed-memory setting, if the consumed data does not reside on local memory, then the producer and consumer of the data must establish communication channels and perform communication. For a task in D2P, producers are the *tasks* mentioned in its input dependences. However, the consumers are the *processes* mentioned in its output dependences. These dependences are computed during inspector’s execution (Figure 3, II.2.d and III.2.e). This computation is dependent on the ability of processes to figure out the owner of a tile (data region) given a tile number and a mapping scheme. E.g. in Figure 5(a) task t_{12} writes to tile 10 and reads from tiles 5 and 9. when t_{12} is being created, last tasks to have computed tile 5 and tile 9 are t_3 and t_{11} respectively. The last task information is available to process P_2 (owner of tile 10, and hence, task t_{12})—and all the processes—due to the SPMD code updating map M (Figure 3, II.2.c). Therefore, process P_2 ’s inspector updates task t_{12} ’s input dependences to expect data from tasks t_3 and t_{11} . The inspector of process P_1 (owner of tiles 5 and 9) updates the output dependences of tasks t_3 and t_{11} to send their results to P_2 .

Once the inspector establishes communication channels between producers and consumers, the actual data movement follows during

executor’s execution of the tasks. Since in D2P, the producers of the data push the data to consumers, consumer process IDs are essential for the tasks to correctly perform communication. A consumer process delivers the data received for consumption to its task(s) based on the input dependences of the task(s). The data is delivered to the correct task if its input dependences contain a task ID that matches the task ID mentioned in the metadata received along with the data (Figure 4, II.3 and II.2.b).

Leaf-task implementation. One advantage of D2P is that end users can rely on D2P for orchestrating inter-task parallelism as shown in Figure 5 while continuing to use their highly-optimized, single-process code within a task. Because a task computes a recursive method with inclusive dependences, users can replace the method body with codes offloading computation to multiple threads, vectorizing processors, GPUs etc. This decoupling of a task’s implementation is safe because of the following: if sub-parts (data computed by an optimized iterative code or spawned sub-tasks) of two coarse-grain tasks overlap, then the inputs to the coarse-grain tasks must overlap because of the inclusion property. As a result, there would be a dependency between the coarse-grain tasks. And since the coarse-grain tasks execute atomically, we are guaranteed to wait for all dependences to be satisfied. Thus, *no two tasks that could interfere can possibly execute in parallel.*

4 IMPLEMENTATION

We have implemented D2P as a domain-specific language and compiler that accepts programs specified as in Figure 2(b) and generates the appropriate inspector-executor distributed-memory code¹. Here, we describe some additional details of D2P’s implementation.

Intra-task parallelism. While D2P primarily concerns itself with identifying and exploiting inter-task parallelism, as shown in Figure 5(b), each task itself represents a fairly coarse-grained computation. D2P leverages the resulting *intra*-task parallelism—easily exposed due to the recursive nature of the implementation—by using Cilk `spawn` [13] to implement parallel sections within a leaf-method body (though, as mentioned in Section 3, the user can instead replace a leaf-method recursive implementation with their own optimized single-process code).

While executing a recursive implementation of a leaf-method, spawning sub-tasks down till the terminating case can greatly increase the parallel overhead. The D2P Codegen generates two versions of the method body (with and without Cilk keywords) and the executor executes the sequential version (without Cilk keywords) to control this overhead. The executor stops executing the parallel version beyond C levels of recursion. We set the spawn-cutoff C to two by default, since, this created sufficient number of sub-tasks for all the benchmarks and the input sizes considered.

Cilk workers (pthreads analogue) execute the spawned sub-tasks. The number of Cilk workers per process reflect a tradeoff between shared-memory parallelism vs. communication-computation overlap: we have two options to subscribe to a fixed number of parallel hardware contexts (cores)—(1) use single Cilk worker per process and more processes or (2) multiple Cilk workers per process

and lesser number of processes. With option 1, there is more fine-grained parallelism since the inspector unrolls more with more processes. The increased fine-grained parallelism provides more opportunities to overlap increased communication with parallel computation. With option 2, lesser processes enforce shallower unrolling, which makes a task compute a larger tile. Larger tiles provide more opportunities to map computation to multiple Cilk workers more effectively. However, as the example on effect inter-section discussed in Section 3 showed, increased task granularity imposes artificial constraints on independent sub-tasks that belong to distinct tasks (Cilk workers in D2P do not communicate with workers from other tasks). Hence, there are lesser opportunities for overlapping communication with computation. This may be acceptable for compute-heavy applications, where communication is not a significant factor. However, for memory-bound applications, we prefer having more fine-grained parallelism. We set the number of Cilk workers to 1 by default and provide a knob to set the Cilk workers. Cilk workers in D2P are not involved in data exchange since the granularity of data communicated is a tile and Cilk workers compute sub-tiles. The process owning a leaf-task sends a tile of data to dependents only when all the Cilk workers executing sub-tasks of the leaf-task finish executing.

Parameters - inspector. We provide blocked, block-cyclic, and hybrid strategies for mapping of data regions to processes. In case of many DP problems, we observe that the data dependences exist along the rows and columns of the DP table, and parallelism is along the diagonal (MWT) or anti-diagonal. Hence, we decompose the problem until the number of tiles in a column (or row) are equal to the number of processes. This avoids column (or row)-wise communication, while still assigning tasks computing tiles along diagonals and anti-diagonals to different processes. As a result, this strategy reduces communication while maintaining parallelism for many DP problems. This strategy also defines the base case trigger in the inspector (Figure 3, II.2.a): the unroll depth D , which hierarchically decomposes a problem (hence, creating the tiles), is dependent on the number of processes. The tiles can only be created when input sizes are known (Figure 3, III.1-size). Hence, the mapping scheme, number of processes, and the input size are the required parameters for the inspector.

Data region IDs. Data ownership is determined based on region IDs in Figure 3 and 4. As region IDs are parameters to an inspector method, the inspector needs to compute them to pass as argument values to a method call (this is done Figure 3, II.3.a). The computation is based on the dimensionality of the matrix computed (DIM) and the quadrant number ($quadrant$). We extract DIM and $quadrant$ from an argument name to a call. E.g. $DIM=2$ (must be the same for all argument names) and $quadrant=1$ for $X01$ in $A(X01, X00, X11)$. Region IDs in the method body are computed as: $(inRegionID * 2^{DIM} + quadrant)$, where $inRegionID$ is the argument value passed to the method call. E.g. the wrapper call to the top-level inspector method $A(box, nullptr, 0, 0)$ always assigns an ID of 0 to the whole data region (Figure 3, III.1). The recursive call to $A(< X11 >)$ within method A in the spec is translated to a call to $A(box11, nullptr, 3, 1)$, where 3 is the new ID computed based on the incoming ID=0 ($box11$ is computed based on box , and $callStackDepth=1$ is incremented for every recursive call). The next

¹Available at <https://bitbucket.org/plcl/d2p>

Method	Description
BaseCase	Recursion terminating case. Specifies the exact function applied in computing a DP table cell. E.g. MWT computes $table_{i,j} = \min(table_{i,j}, table_{i,k} + table_{k,j} + weight_{i,k,j})$.
InitTable	Captures algorithm-specific initialization of the DP table. E.g. MWT initializes all the elements on the main diagonal to zero.
GetScore	Fetches the optimal score. Aggregate functions MAX, MIN are provided to fetch the maximum/minimum value among the distributed DP table cells. Specific cell content can be obtained using <code>GetCell(i,j)</code> . In MWT, <code>GetCell(0,N-1)</code> fetches the optimal score, since the top-right corner cell always stores the optimal score.
ReadInput	Reads input-data. E.g. MWT reads 2D coordinates of points from a file

Table 1: Auxiliary methods in D2P.

recursive call to $A(< X11 >)$ translates to $A(box11, nullptr, 15, 2)$. As a result, the tiles are assigned IDs in Z-order [29]. Given a Z-order number, the corresponding grid coordinates can be efficiently computed. Based on the grid coordinates and the mapping scheme (e.g. alternating columns of tiles to different processes), tile owners can be computed.

Auxiliary methods. The base case was explained earlier in Section 2. Table 1 describes other auxiliary methods, which capture application-specific properties such as initializing the DP table, fetching the optimal score computed, and reading inputs. As these methods are free from the concerns of distribution, we require end-users to implement them and augment the generated code. We also provide their reference implementations for five benchmarks.

Autogen. Writing a specification such as the one for the MWT problem shown in Figure 2(b) is not straightforward. Autogen is a tool that automatically discovers recursive DP algorithms for a subset of DP problems belonging to the *Fractal-DP* class [10]. Autogen takes as input an iterative code snippet consisting of only the `for` loop structure (similar to lines [3-4,7-8] in Figure 2(a)), and produces a pseudocode similar to that of Figure 2(b), but with additional `parallel` annotations describing method calls that can be executed in parallel. The Autogen paper has more details on how Autogen infers recursive methods and extracts parallelism.

We design the D2P Codegen to parse Autogen’s output. Thereby, we create an end-to-end system for generating distributed-memory codes for a wide variety of DP problems. As a result, an end-user can generate distributed-memory code for a real-world application by translating only the embedded loop structure representing the DP table recurrence equation. Currently we rely on Autogen’s `parallel` annotations in its output to exploit intra-task parallelism. We could extend our inter-task parallelism extraction scheme to identify parallelism within a method body in future. Note that we use Autogen primarily as a tool to generate specifications for D2P. In the next Section, we evaluate D2P with a specification that Autogen cannot generate.

5 EVALUATION

We present the evaluation results of D2P with five DP based algorithms. These algorithms are drawn from the domains of bio-informatics, computational geometry, and computer science. The algorithms differ in their data dependency pattern and are spread over the different categories of DP problems mentioned in Galil et al. [19]. We present the scalability results and show case studies comparing D2P implementation of a benchmark with i) DPX10 [40], a generic framework for implementing distributed-memory DP algorithms and ii) a hand-written distributed-memory program [39].

Benchmarks.

- i **Minimum Weight Triangulation (MWT)** [25] is a triangulation algorithm, detailed in Section 2.
- ii **Matrix Chain Multiplication (MCM)** [14] finds the optimal way to associate a sequence of matrix multiplications. Computing a single DP table cell requires reading from $O(N)$ cells in both MWT and MCM, and they compute only the upper triangular matrix of the DP table.
- iii **Smith-Waterman Local Alignment (SW)** [35] determines the similarity of two DNA (or amino acid) sequences. All pairs of possible *subsequences* from both the sequences are compared and scored rather than considering whole sequences. The algorithm finds local regions within the sequences having an optimal similarity score. In global sequence alignment algorithms such as Needleman-Wunsch (NW), the whole sequence is considered. The recursive formulations for both SW and NW are the same and the implementation differs only in the terminating case. Hence, we evaluate only SW for scalability.
- iv **Floyd-Warshall All Pairs Shortest Path (APSP)** [17] finds the shortest path between every pair of vertices in a graph. APSP computes the entire DP table matrix. SW and APSP are instances of the *gap* problem [19] where, SW reads $O(1)$ and APSP reads $O(N)$ cells to compute a cell.
- v **RNA Problem (RNA)** [31] predicts the structure produced as an RNA strand folds onto itself. Our RNA implementation stores similarity values of all possible pair of subsequences to account for multi-loops, leading to a 4-dimensional DP table. Computing a DP table cell requires reading $O(1)$ cells.

We use both synthetic and real-world data sets [8, 34]. Among synthetic data sets, MWT uses a convex hull of randomly generated points in 2D space, and MCM uses randomly generated integer weights in the range 0 to 1000. SW uses Ecoli sequences NC_000913.2E and BA000007.2, and RNA uses mRNA sequence of fruit-fly [34]. APSP uses Autonomous Systems (AS) relationships graph from the ISP data set [8].

5.1 Methodology

The distributed-memory (DM) implementations of DP algorithms are based on the Autogen produced shared-memory pseudocodes for all the benchmarks except MCM. In case of MCM, Autogen fails to discover a recursive algorithm since the MCM recurrence does not belong to the *Fractal-DP* class. So, we wrote the specification for MCM with hints from the MWT specification, as both MWT and MCM belong to a similar category of DP problems. Our baselines are shared-memory implementations of recursive DP algorithms. Being recursive implementations, they are optimized for locality and represent the best single process sequential implementations of each benchmark among the approaches we tested. A comparison with sequential iterative implementations for a subset of the benchmarks confirmed this: our recursive baselines (*1_rec*) of MCM and MWT were 1.08× and 1.36× faster than their respective iterative sequential implementations. We assume that the results hold for remaining benchmarks based on the findings in the Autogen paper.

The overall computation in a DP application can be divided into three steps: 1) table initialization, 2) table computation, and 3) backtracking. Backtracking is an important step constructing

the optimal solution based on the optimal scores computed in step 2. As step 2 is computationally dominant in a majority of the DP problems, step 2 alone is timed in all performance measurements unless otherwise noted. The runtimes are measured using wall-clock time and every configuration of a test is run until a steady state is achieved.

Development and execution environment. The DM implementations of all benchmarks use C++11, Intel MPI [23], and Intel CilkPlus [13]. We compile the programs with `mpiicpc`, a wrapper compiler for ICC 16.0.3. The experiments are run on Bridges, an XSEDE [38] cluster of over 752 nodes having Intel-OPA interconnect. We accessed up to 32 nodes of this cluster. Each node of the cluster runs CentOS Linux 7.4.1708, contains two 14-core Intel-E5-2695v3 processors with 128GB main memory.

5.2 Scalability

Strong scaling. In these experiments, we measure the performance of the benchmarks with increasing number of computing resources (processes, Cilk workers) for a fixed input size. The input sizes are chosen such that slowest DM executions complete in a reasonable amount of time.

Figure 6 shows strong-scaling with the default unrolling strategy (D based on number of processes) and 1 Cilk worker per process. The Figure also shows strong-scaling with the best value of D determined empirically for each process configuration. As single-process runs do not unroll recursion, best D is not applicable and hence, is not shown. The plots show speedup of a DM execution over the sequential recursive baseline (1_{rec}). In scenarios when input sizes prohibit allocating the DP table in a contiguous heap-space, we use a two-process DM run (2_{DM}) as the baseline.

As the results show, D2P implementations scale well given the inherently strong data dependencies in DP problems. We also observe that D2P’s default unrolling strategy is sufficient. The speedup curve with default D closely follows the best D in most cases. As unrolling is commensurate with the number of processes, at larger number of processes, there is more unrolling. Deeper unrolling (in search of better D) at large number of processes causes an increase in preprocessing overhead, which outweighs the benefits obtained from increased available parallelism. As a result, best D eventually matches the default D . Since RNA uses a 4D table, with every level unrolled, there is a 16 $\times$ increase in the DP table tiles. So, the best and the default D are same from level 2 onwards. Hence, we do not see an observable difference in performance.

We study the effect of intra-task parallelism with multiple Cilk workers per process. Table 2 shows the speedups obtained with multiple Cilk workers and compares them with the best speedup numbers shown in Figure 6. The maximum number of cores used is the same across both single and multiple Cilk worker configurations. We performed a sweep of 1 to 16 Cilk workers per process at each scale, however, we show here only the best numbers (W), which were 16 Cilk workers for up to 64 processes and 8, 4, and 2 Cilk workers for 128, 256, and 512 processes respectively. Figure 7 summarizes the results and shows the configurations that yield these runtimes.

Our experiments showed that allocating parallelism to additional processes rather than additional Cilk workers consistently gave better performance in some benchmarks, even though using more processes requires more communication. This is a consequence of the tradeoff between shared-memory parallelism vs. communication-computation overlap mentioned earlier (Section 4, intra-task parallelism). MWT, MCM, and APSP are compute-bound and hence, due to better utilization of available shared-memory parallelism, multiple Cilk workers yield almost double the speedups (280.1 $\times$, 218.2 $\times$, and 90.3 $\times$ resp.) compared to single Cilk worker runs. Additional processes rather than additional Cilk workers yield better performance in memory-bound benchmarks (SW and RNA). We also measure memory contention and find that the intra-node overheads associated with communication do not play a role in memory-bound benchmarks not seeing improved speedups with multiple Cilk workers. We observed that the contention *decreased* across all benchmarks—memory- and compute-bound—, since adding more Cilk workers meant reducing the number of processes and hence, the number of communicating entities per node.

Overall, we see an 89.6 $\times$ geomean speedup in the default configuration, which further increases to 125.4 $\times$ when we exploit intra-task parallelism. The addition of more Cilk workers to exploit intra-process parallelism may be necessary in order to get the absolute best performance. The results reflect D2P’s use of coarse-grained, task-level parallelism at the level of leaves of the tree and Cilk workers to exploit an intra-process, sub-task parallelism available within a task.

Weak Scaling. In these experiments, we increase the input size with increasing number of processes while keeping the per-process computation fixed. We also override the default D2P configuration and keep the unroll depth D (problem decomposition) constant. As a result, the runtime sometimes decreases with more processes due to more effective utilization of available parallelism. In weak-scaling experiments in general, we expect the communication overhead to increase proportional to the number of processes. In D2P benchmarks, we have an additional source of communication overhead: as some benchmarks read $O(N)$ (APSP, MWT, MCM) data to compute a cell, the amount of communication *per cell* can increase with scale. Additionally, when the input size is not a perfect power-of-two the recursive decomposition does not evenly divide the table, so partitioning tasks/sub-tasks computing these tiles can result in load-imbalance. As a result, we expect the overall runtime to increase with scaling up inputs and processes.

Figure 8 confirms this behavior. We see that the runtimes show different degrees of increase for different benchmarks. The 8 process run in SW uses an input sequence length of 16,384, which is perfect-power-of-two. Because SW is $O(N^2)$ algorithm, the next perfect-power-of-two input size is used when the processes are scaled to 32. We consider scaling from 8-32-128-512 as an indicator of true weak-scaling since these correspond to perfect load-balance. For MWT and APSP, $O(N^3)$ algorithms, 8-64-512 process runs represent the load-balanced configurations. In RNA, which is $O(N^4)$, 32 and 512 process runs are each perfectly balanced.

In a perfectly load-balanced scenario, SW, MWT, APSP, and RNA weak-scale differently. SW weak-scales the best and shows a flat scaling due to a $O(1)$ amount of communication per cell.

Figure 6: Strong scaling in D2P benchmarks showing speedup of single Cilk worker/process configuration over baselines. SW and RNA use baseline runtimes normalized w.r.t. 2-process (2_{DM}) run instead of using the 1_{rec} baseline runtimes. Also shown is a comparison of speedups obtained with the default unrolling depth, and the best unrolling depth empirically measured.

Processes	MCM			MWT			SW			APSP			RNA		
1_{rec}	1025.4s			1560.8s			NA			4570.6s			NA		
	Pre	Cilk-workers		Pre	Cilk-workers		Pre	Cilk-workers		Pre	Cilk-workers		Pre	Cilk-workers	
		1	W		1	W		1	W		1	W		1	W
1_{DM}	0.11s	0.9×	7.2×	0.1s	0.9×	4.4×	NA	NA	NA	12.4s	0.9×	7.8×	NA	NA	NA
2_{DM}	0.05s	1.2×	10.4×	0.04s	1.2×	7.3×	8.2s	1×	3.2×	6.1s	1.6×	17.1×	10.5s	1×	6.6×
4_{DM}	0.02s	1.8×	17.4×	0.02s	1.7×	14.8×	4.4s	1.5×	2.9×	3.4s	2.9×	28.9×	3.2s	1.9×	7.8×
8_{DM}	0.01s	3.6×	38.8×	0.01s	3.5×	33.8×	2.3s	2.8×	3.9×	1.8s	5.4×	46.8×	3.2s	2.7×	9.5×
16_{DM}	0.01s	8.5×	75.9×	0.01s	7.5×	75.4×	1.2s	5.3×	7.1×	0.98s	9.2×	55.3×	4.7s	3.1×	7.1×
32_{DM}	0.01s	16.9×	96.7×	0.02s	15.8×	63.2×	0.9s	9.4×	7.2×	0.55s	14.8×	58.8×	3.3s	4.5×	12.2×
64_{DM}	0.02s	23.3×	160.2×	0.2s	25.2×	108.4×	0.6s	19.1×	10.2×	0.68s	16.6×	49.4×	2.5s	6.8×	12.5×
128_{DM}	0.08s	36.1×	218.2×	0.28s	44.2×	289.1×	0.5s	28.9×	12.6×	0.42s	17.8×	40.4×	2s	7.8×	13.6×
256_{DM}	0.58s	84.8×	183.1×	0.04s	86.2×	124.8×	0.4s	38.6×	16.2×	0.27s	22.7×	30.2×	1.7s	9.9×	11.9×
512_{DM}	0.75s	153.1×	63.6×	0.13s	160.9×	84.4×	0.9s	61.1×	8.2×	1.8s	60.3×	23.1×	1.5s	14.2×	3.8×

Table 2: Details of (i) baseline (1_{rec}) and preprocessing overhead (Pre) runtimes in seconds and (ii) Comparison of speedups ($\times$) obtained with 1 and W Cilk workers per process over baseline runtimes. The single Cilk worker per process numbers are from Figure 6. W represents the workers corresponding to the best run obtained from a sweep of 1 to 16 Cilk workers.

Figure 7: Summarizing Table 2 results. Exploiting intra-task parallelism is necessary in compute-bound benchmarks.

Even though MWT and APSP have both $O(N)$ dependencies in computing a cell, MWT shows 1.7 $\times$ increase while APSP shows a 2.8 $\times$ increase in runtime. This is because the communication overhead is reduced in MWT, since, it computes only the upper triangular matrix. RNA, despite computing $O(N^4)$ cells, shows a modest 1.3 $\times$ increase in runtime. This is because, like SW, each cell in RNA depends on $O(1)$ other cells.

Preprocessing overheads and Backtracking. the preprocessing time due to unrolling, dependency analysis, and partitioning remained within 1.7% of the DP table computation time in most cases. Note that all our performance numbers include this preprocessing time. We also implement backtracking manually in the D2P generated code and observe that for representative benchmarks (MWT, SW, and APSP), backtracking time at any scale remains within 2.1% of the DP table computation time, thus highlighting that table computation still dominates other steps.

5.3 Case Studies

D2P vs. Hand_Written. We compare SW implementation of D2P with UPC, a highly optimized, hand-written, distributed-memory

implementation of iterative SW algorithm [39]. UPC takes a static approach to task partitioning and communication. The master-slave based implementation in UPC divides the DP table (row-wise) evenly among slaves and assigns any undivided (remaining) rows to the master. As a result, a process depends on only a single row of data from one other process owning the adjacent DP table row above. In contrast, note that a producer in D2P sends an entire tile of data to the consumer. The slaves communicate their results back to the master, who computes any assigned rows based on the results received. A pipeline-parallel scheme divides the rows further into blocks and helps exploit intra-process parallelism.

Figure 9(a) shows strong- and weak-scaling results. These measurements were obtained with 95% confidence interval with 0.08% margin of error. We chose the maximum input size permissible in UPC. Since UPC adopts a data-parallel approach, each process in UPC can only allocate a DPTable of a maximum size of 4GB in heap and this corresponds to an input size of 32768 bases. We observe that D2P scales better than UPC. Because of the default terminating case computing a single cell and the memory-bound nature of SW requiring deeper unrolling (Section 5.2), D2P generated code is slower at smaller number of processes. At larger number of processes, D2P outperforms UPC due to aggressive unrolling.

D2P vs. DPX10 [40]. DPX10 is the best available framework for parallelizing iterative DP programs on distributed-memory systems. Like D2P, DPX10 exposes a set of APIs to the end-user, who specifies existing parallelism pattern and implements the iterative algorithm. Also like D2P, DPX10 captures node-level and thread-level parallelism, however, through X10 Places and pthreads respectively.

We compare the performance of SW benchmark only, since, DPX10 currently supports only DP programs with 2-dimensional DP table and $O(1)$ dependency in computing a cell (2D/0D) [40].

Figure 8: Weak scaling in D2P benchmarks. Y-axis shows the normalized runtime w.r.t. the 8-process run. The data-labels show the input size used for that execution.

Figure 9: Case studies comparing D2P with other systems.

DPX10 takes a dynamic approach to task management through a master-slave implementation: first, a master process creates a DAG of tasks based on the parallelism pattern. The tasks are then distributed among slaves, who report back to the master with results. The master and every slave coordinate task execution with the help of an additional per-slave (and master) scheduler processes.

Figure 9(b) shows the strong scaling results. We show the best performing DPX10 run among different choices of X10 threads per place. DPX10 runs timed-out at larger scales and hence, are not shown. Here, D2P is significantly better (up to 412 $\times$ speedup when best runs are compared) than DPX10 in raw performance numbers as well as scalability. We believe that the design and implementation choices (dynamic partitioning and execution of tasks, dedicated schedulers) in DPX10 affect its performance. Overall, the design choices (owner-computes, finalizing the partitioning of tasks before their execution) help D2P outperform DPX10.

6 RELATED WORK

Distributed-memory parallel programs and simplifying their creation has been extensively studied for many decades with a focus on iterative formulations [3, 6, 7, 9, 22, 26, 32, 33]. Recent research has shown that recursive formulations have good locality properties, readily expose parallelism and hence, can be as effective as iterative codes if implemented carefully [10, 11, 27]. In D2P we identify additional properties that, when true for recursive formulations, simplify the creation of distributed-memory parallel programs. Lifflander *et al.* [27] and Bauer *et al.* [4] (Legion) use programmer specified ‘effect’ annotations to identify dependences and extract parallelism. The effect annotations are explicit way of specifying inclusive properties for a recursive method. D2P’s unrolling and analysis generates the required annotations automatically. In [27], different recursive method calls are spliced (interleaved) effectively to improve the cache performance of recursive implementations of stencil computations on shared-memory (SHM)

systems. In Legion, the annotated code is translated and the computation is mapped to a heterogenous system. To our knowledge, no other auto-parallelization tool except Legion handles recursive codes for mapping computation to distributed-memory (DM) systems. However, as Legion targets advanced programmers, it requires significant tuning. Because of the inclusive nature of data accesses in recursive formulations, effect annotations are redundant in divide-conquer algorithms with the inclusive property. However, we could use annotations to extend our code generation scheme to a broader set of recursive programs without the necessary properties.

D2P differs from existing general-purpose code generation frameworks in a few ways: D2P extracts parallelism (via unrolling), does explicit partitioning, and targets recursive divide-conquer formulations with inclusive and intersection properties. While D2P does static allocation of work (completed when inspector finishes executing at runtime), others do a runtime task management [4, 7, 22, 40]. Some frameworks [3, 26] rely on explicit specification of parallelism (e.g. `#pragma omp parallel for`) and implicit partitioning (implicit assignment of iterations to threads). While others [32, 33, 37] do not assume this information and work well on regular/irregular/mixed iterative codes. Sarkar *et al.* [33] automate program partitioning in a functional programming setting, and explore compile- and runtime scheduling techniques on multi-processors with different system architectures. Unlike these fully automated systems that translate a general SHM program to a DM program, D2P is a semi-automatic framework that translates a specification of a recursive divide-conquer algorithm to a distributed-memory program. Semi-automatic because the specification can be incomplete, thus requiring end-users to augment D2P produced code. However, D2P is not bound by any assumptions (e.g. about the maximum input size to produce a data-parallel code in [9, 26, 39]).

Dynamic programming [5] algorithms, with their application in a wide variety of domains, are the target of locality, parallelizing, and communication optimizations [1, 2, 10, 12, 16, 19, 21, 28, 40, 41] on both SHM and DM systems. FastFlow [1], EasyHPS [16] and its

successor DPX10 [40] specialize on iterative DP programs for DM systems. Program synthesis approaches [24, 36] to dynamic programs take a spec (e.g. a partial SHM program) with some omissions (holes) and fill it to produce a complete program. D2P, like these systems, works on a spec. However, the spec is high-level outline of a recursive algorithm and omits application-specific details. The omission is expected to be filled in the D2P produced code by the end-user. The inspector-executor approach of Saltz *et al.* is used in D2P. We could augment the partitioning scheme in D2P with sophisticated inspector-executor based techniques [20, 32, 33, 37].

Galil *et al.* [19] categorize DP algorithms based on the dependency patterns and design SHM parallel iterative algorithms. Chowdhury *et al.* [10–12] design SHM parallel recursive DP algorithms and automate their design too. In addition, they show that SHM parallel recursive DP implementations adapt better in the presence of cache sharing, are an order of magnitude faster, and have more predictable runtimes than their tiled iterative counterparts [10]. The parallel DP formulations (both iterative and recursive) previously described perform an exact computation of the DP table. In contrast, Maleki *et al.* take an approximate-computing based approach to parallelize DP table computation on both SHM and DM systems. They use the insight that for a narrower set of DP algorithms having the linear-tropical-dynamic-programming (LTDP) property [28], where cells depend only on a previous wavefront, a cell computation need not wait for all dependences prescribed in the exact computation to be satisfied. D2P does exact computation of the DP table. As a result, it can parallelize parenthesis, APSP, and RNA problems, which have dependencies on all the previous wavefronts. Majority DM parallelizing schemes based on exact computation of SW [15, 30] exploit an application-level, coarse-grained parallelism available. They compare a sequence against a database of sequences. The SW implementation of D2P, in contrast, compares (and parallelizes) two sequences as required by the SW algorithm.

7 CONCLUSIONS

In this paper, we presented D2P, a system for auto-generating distributed-memory implementations of recursive divide-conquer algorithms with inclusive and intersection properties. We showed how D2P can be used to generate those implementations from specifications. We also showed how D2P generates tasks, partitions them, and inserts communication. Finally, we evaluated D2P with recursive formulations of DP algorithms showing that the generated implementations scale well, overheads are negligible, run significantly faster than those created manually using a similar framework, and even outperform optimized hand-written codes in most cases.

8 ACKNOWLEDGMENTS

The authors would like to thank Rezaul Chowdhury for providing the source code of Autogen, Samuel Midkiff for providing access to the Wabash cluster used during the development of D2P, and anonymous referees for their helpful comments and suggestions. This work was supported in part by the following grants: CCF-1150013, DOE DE-SC0010295, and XSEDE TG-ASC170007.

REFERENCES

- [1] Marco Aldinucci, Massimiliano Meneghin, and Massimo Torquati. 2010. Efficient Smith-Waterman on multi-core with FastFlow. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*. IEEE, 195–199.
- [2] Matthew Baker, Aaron Welch, and Manjunath Gorenthla Venkata. 2014. Parallelizing the Smith-Waterman algorithm using OpenSHMEM and MPI-3 one-sided interfaces. In *Workshop on OpenSHMEM and Related Technologies*. Springer, 178–191.
- [3] Ayon Basumallik and Rudolf Eigenmann. 2006. Optimizing irregular shared-memory applications for distributed-memory systems. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 119–128.
- [4] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2012. Legion: Expressing locality and independence with logical regions. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 1–11.
- [5] Richard Bellman. 1957. *Dynamic Programming* (1 ed.). Princeton University Press, Princeton, NJ, USA. <http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false>
- [6] Uday Bondhugula. 2011. *Automatic Distributed Memory Code Generation using the Polyhedral Framework*. Technical Report IISc-CSA-TR-2011-3. Indian Institute of Science, Bangalore.
- [7] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Hérault, and Jack J Dongarra. 2013. Parsec: Exploiting heterogeneity to enhance scalability. *Computing in Science & Engineering* 15, 6 (2013), 36–45.
- [8] caida07 [n. d.]. The CAIDA AS Relationships Dataset, Nov'5, 2007. <http://www.caida.org/data/as-relationships/>. ([n. d.]).
- [9] Sanjay Chatterjee, Sagnak Tasirlar, Zoran Budimlic, Vincent Cave, Milind Chabbi, Max Grossman, Vivek Sarkar, and Yonghong Yan. 2013. Integrating asynchronous task parallelism with MPI. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 712–725.
- [10] Rezaul Chowdhury, Pramod Ganapathi, Jesmin Jahan Tithi, Charles Bachmeier, Bradley C Kuszmaul, Charles E Leiserson, Armando Solar-Lezama, and Yuan Tang. 2016. Autogen: Automatic discovery of cache-oblivious parallel recursive algorithms for solving dynamic programs. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 10.
- [11] Rezaul Alan Chowdhury, Hai-Son Le, and Vijaya Ramachandran. 2010. Cache-oblivious dynamic programming for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 7, 3 (2010), 495–510.
- [12] Rezaul Alan Chowdhury and Vijaya Ramachandran. 2008. Cache-efficient dynamic programming algorithms for multicores. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*. ACM, 207–216.
- [13] CilkPlus [n. d.]. Intel Cilk Plus. <https://www.cilkplus.org/>. ([n. d.]).
- [14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.
- [15] Aaron E Darling, Lucas Carey, and W Chun Feng. 2003. *The design, implementation, and evaluation of mpiBLAST*. Technical Report. Los Alamos National Laboratory.
- [16] Jun Du, Ce Yu, Jizhou Sun, Chao Sun, Shanjia Tang, and Yanlong Yin. 2013. EasyHPS: A multilevel hybrid parallel system for dynamic programming. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 630–639.
- [17] Robert W Floyd. 1962. Algorithm 97: shortest path. *Commun. ACM* 5, 6 (1962), 345.
- [18] Matteo Frigo, Charles E Leiserson, and Keith H Randall. 1998. The implementation of the Cilk-5 multithreaded language. In *ACM Sigplan Notices*, Vol. 33. ACM, 212–223.
- [19] Zvi Galil and Kunsoo Park. 1994. Parallel algorithms for dynamic programming recurrences with more than O(1) dependency. *J. Parallel and Distrib. Comput.* 21, 2 (1994), 213–222.
- [20] Kang Su Gatlin and Larry Carter. 1999. Architecture-cognizant divide and conquer algorithms. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*. ACM, 25.
- [21] Khaled Hamidouche, Fernando Machado Mendonca, Joel Falcou, Alba Cristina Magalhaes Alves de Melo, and Daniel Etienne. 2013. Parallel Smith-Waterman Comparison on Multicore and Manycore Computing Platforms with BSP++. *International Journal of Parallel Programming* 41, 1 (01 Feb 2013), 111–136. <https://doi.org/10.1007/s10766-012-0209-6>
- [22] Chao Huang and Laxmikant Kale. 2007. Charisma: Orchestrating migratable parallel objects. In *Proceedings of the 16th international symposium on High performance distributed computing*. ACM, 75–84.
- [23] intelmpi [n. d.]. Intel MPI Libraries. <https://software.intel.com/en-us/intel-mpi-library>. ([n. d.]).
- [24] Shachar Itzhaky, Rohit Singh, Armando Solar-Lezama, Kuat Yessenov, Yongquan Lu, Charles Leiserson, and Rezaul Chowdhury. 2016. Deriving divide-and-conquer

- dynamic programming algorithms using solver-aided transformations. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM, 145–164.
- [25] GT Klincsek. 1980. Minimal triangulations of polygonal domains. *Annals of Discrete Mathematics* 9 (1980), 121–123.
- [26] Okwan Kwon, Fahed Jubair, Rudolf Eigenmann, and Samuel Midkiff. 2012. A hybrid approach of OpenMP for clusters. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 75–84.
- [27] Jonathan Lifflander and Sriram Krishnamoorthy. 2017. Cache locality optimization for recursive programs. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 1–16.
- [28] Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2014. Parallelizing dynamic programming through rank convergence. *ACM SIGPLAN Notices* 49, 8 (2014), 219–232.
- [29] Guy M Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. (1966).
- [30] Mahdi Noorian, Hamidreza Pooshfam, Zeinab Noorian, and Rosni Abdullah. 2009. Performance enhancement of smith-waterman algorithm using hybrid model: Comparing the mpi and hybrid programming paradigm on smp clusters. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 492–497.
- [31] Ruth Nussinov, George Pieczenik, Jerrold R Griggs, and Daniel J Kleitman. 1978. Algorithms for loop matchings. *SIAM Journal on Applied mathematics* 35, 1 (1978), 68–82.
- [32] Mahesh Ravishankar, Roshan Dathathri, Venmugil Elango, Louis-Noël Pouchet, J Ramanujam, Atanas Rountev, and P Sadayappan. 2015. Distributed memory code generation for mixed irregular/regular computations. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 65–75.
- [33] Vivek Sarkar. 1989. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA.
- [34] Stephen T Sherry, M-H Ward, M Kholodov, J Baker, Lon Phan, Elizabeth M Smigielski, and Karl Sirotkin. 2001. dbSNP: the NCBI database of genetic variation. *Nucleic acids research* 29, 1 (2001), 308–311.
- [35] T.F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195 – 197. [https://doi.org/10.1016/0022-2836\(81\)90087-5](https://doi.org/10.1016/0022-2836(81)90087-5)
- [36] Saurabh Srivastava, Sumit Gulwani, and Jeffrey S Foster. 2010. From program verification to program synthesis. In *ACM Sigplan Notices*, Vol. 45. ACM, 313–326.
- [37] Michelle Mills Strout, Alan LaMielle, Larry Carter, Jeanne Ferrante, Barbara Kreaseck, and Catherine Olschanowsky. 2016. An Approach for Code Generation in the Sparse Polyhedral Framework. *Parallel Comput.* 53, C (April 2016), 32–57.
- [38] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 5 (Sept.-Oct. 2014), 62–74. <https://doi.org/10.1109/MCSE.2014.80>
- [39] upesw [n. d.]. SmithWaterman with OpenMPI and OpenMP. <https://www.alexjf.net/projects/distributed-systems/smith-waterman-openmp-and-openmpi/>. ([n. d.]).
- [40] Chen Wang, Ce Yu, Shanjiang Tang, Jian Xiao, Jizhou Sun, and Xiangfei Meng. 2016. A general and fast distributed system for large-scale dynamic programming applications. *Parallel Comput.* 60 (2016), 1–21.
- [41] Wenduo Zhou and David K Lowenthal. 2006. A parallel, out-of-core algorithm for RNA secondary structure prediction. In *Parallel Processing, 2006. ICPP 2006. International Conference on*. IEEE, 74–81.
- [42] Michal Ziv-Ukelson, Irit Gat-Viks, Ydo Wexler, and Ron Shamir. 2008. A faster algorithm for RNA co-folding. In *International Workshop on Algorithms in Bioinformatics*. Springer, 174–185.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

The distributed-memory implementations of all benchmarks use C++11, Intel MPI, and Intel CilkPlus. The programs are compiled with mpiicpc, a wrapper compiler for ICC 19.0.3. The experiments are run on Bridges (Bridges-Regular Memory), an XSEDE cluster. The recursive shared-memory implementations used as baseline are implemented in C++ and compiled using ICC 19.0.3 tool chain. The following software are used: 1) Autogen to generate inputs to our system. 2) UPC_SW: A handwritten distributed-memory implementation of Smith-Waterman algorithm freely available at: <https://www.alexjf.net/projects/distributed-systems/smith-waterman-openmp-and-openmpi/> 3) DPX10: A distributed-memory framework for parallelizing iterative DP codes. Freely available at <https://github.com/wangvsa/DPX10> All these applications are run on Bridges cluster to support our evaluation.

The data sets come from synthetic and real-world sources: 1) Ecoli sequences NC_000913.2E, BA000007.2, and mRNA sequence of fruit fly from NCBI genetic variation database. 2) Autonomous Systems relationships data set from ISP providers: <http://www.caida.org/data/as-relationships/> (for the date 11/5/2007). 3) synthetic inputs a) convex hull of randomly generated points in 2D plane. b) a text file of 16,000 numbers in the range (0,1000) randomly chosen.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: There are no author-created data artifacts.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

List of URLs and/or DOIs where artifacts are available:
<https://bitbucket.org/plcl/d2p/src/master/>

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: HPC Cluster, Bridges, Regular Memory

Operating systems and versions: CentOS Linux 7.4.1708

Compilers and versions: ICC 19.0.3.199

Libraries and versions: Intel MPI that comes with toolchain icc 19.0.3

Paper Modifications: An older version of the D2P software is used to handle input sizes that are non-powers-of-two in RNA weak-scaling experiments shown in Figure 7.

Output from scripts that gathers execution environment information.

```
AUTOJUMP_ERROR_PATH=/home/USER/.local/share/autojump
↪ /errors.log
CPLUS_INCLUDE_PATH=/opt/intel/clck/2019.2.1/include
MKLROOT=/opt/intel/compilers_and_libraries_2019.3.19
↪ 9/linux/mkl
MANPATH=/opt/intel/itac/2019.3.032/man:/opt/intel/cl
↪ ck/2019.2.1/man:/opt/intel/man/common:/opt/intel
↪ /compilers_and_libraries_2019.3.199/linux/mpi/ma
↪ n:/opt/intel/documentation_2019/en/debugger/gdb-
↪ ia/man/:/opt/packages/pgi/linux86-64/19.4/mpi/op
↪ enmpi-3.1.3/share/man:/opt/packages/pgi/linux86-
↪ 64-llvm/19.4/man:/opt/packages/xdusage/2.1-1/sha
↪ re/man:/opt/packages/slurm/default/share/man:/us
↪ r/local/man:/usr/share/man:
XDG_SESSION_ID=97678
HOSTNAME=br006.pvt.bridges.psc.edu
VT_MPI=impi4
AUTOJUMP_SOURCED=1
PGI_OPTL_LIB_DIRS=/opt/packages/pgi/linux86-64/19.4/
↪ mpi/openmpi-3.1.3/lib
INTEL_LICENSE_FILE=/opt/intel/licenses
IPPROOT=/opt/intel/compilers_and_libraries_2019.3.19
↪ 9/linux/ipp
SHELL=/usr/psc/shells/bash
TERM=xterm
HISTSIZE=1000
PROJECT=/pylon2/ac560jp/USER
SSH_CLIEN=128.46.4.86 33678 22
GLOBUS_LOCATION=/usr
LIBRARY_PATH=/opt/intel/clck/2019.2.1/lib/intel64:/o
↪ pt/intel/compilers_and_libraries_2019.3.199/linu
↪ x/mpi/intel64/libfabric/lib:/opt/intel/compilers
↪ _and_libraries_2019.3.199/linux/ipp/lib/intel64:
↪ /opt/intel/compilers_and_libraries_2019.3.199/li
↪ nux/compiler/lib/intel64_lin:/opt/intel/compiler
↪ s_and_libraries_2019.3.199/linux/mkl/lib/intel64
↪ _lin:/opt/intel/compilers_and_libraries_2019.3.1
↪ 99/linux/tbb/lib/intel64/gcc4.7:/opt/intel/comp
↪ lers_and_libraries_2019.3.199/linux/daal/lib/int
↪ el64_lin
PGI_OPTL_INCLUDE_DIRS=/opt/packages/pgi/linux86-64/1
↪ 9.4/mpi/openmpi-3.1.3/include
QTDIR=/usr/lib64/qt-3.3
QTINC=/usr/lib64/qt-3.3/include
SSH_TTY=/dev/pts/124
QT_GRAPHICSSYSTEM_CHECKED=1
USER=USER
```

```

LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0
↪ 1;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;0
↪ 1:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=3
↪ 0;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tg
↪ z=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lh
↪ a=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.t
↪ lz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.z
↪ ip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;
↪ 31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31
↪ :*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
↪ :*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:
↪ *.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:
↪ *.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31
↪ :*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*
↪ .jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:
↪ *.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:
↪ *.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35
↪ :*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;3
↪ 5:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;
↪ 35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01
↪ ;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01
↪ ;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;
↪ 35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01
↪ ;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;3
↪ 5:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;3
↪ 5:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;3
↪ 5:*.aac=01;36:*.au=01;36:*.flac=01;36:*.mid=01;3
↪ 6:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;
↪ 36:*.ogg=01;36:*.ra=01;36:*.wav=01;36:*.axa=01;3
↪ 6:*.oga=01;36:*.spx=01;36:*.xspf=01;36:
LD_LIBRARY_PATH=/opt/intel/itac/2019.3.032/intel64/s
↪ lib:/opt/intel/compilers_and_libraries_2019.3.19
↪ 9/linux/mpi/intel64/libfabric/lib:/opt/intel/com
↪ pilers_and_libraries_2019.3.199/linux/mpi/intel6
↪ 4/lib/release:/opt/intel/compilers_and_libraries
↪ _2019.3.199/linux/mpi/intel64/lib:/opt/intel/com
↪ pilers_and_libraries_2019.3.199/linux/ipp/lib/in
↪ tel64:/opt/intel/compilers_and_libraries_2019.3.
↪ 199/linux/compiler/lib/intel64_lin:/opt/intel/co
↪ mpilers_and_libraries_2019.3.199/linux/mkl/lib/i
↪ ntel64_lin:/opt/intel/compilers_and_libraries_20
↪ 19.3.199/linux/tbb/lib/intel64/gcc4.7:/opt/intel
↪ /debugger_2019/libipt/intel64/lib:/opt/intel/com
↪ pilers_and_libraries_2019.3.199/linux/daal/lib/i
↪ ntel64_lin:/opt/packages/pgi/linux86-64/19.4/mpi
↪ /openmpi-3.1.3/lib:/opt/packages/pgi/linux86-64-
↪ llvm/19.4/lib
PSTLR00T=/opt/intel/compilers_and_libraries_2019.3.1
↪ 99/linux/pstl
SUDO_PROMPT=Password for %u@PSC.EDU
FI_PROVIDER_PATH=/opt/intel/compilers_and_libraries_
↪ 2019.3.199/linux/mpi/intel64/libfabric/lib/prov
SSH_AUTH_SOCK=/tmp/ssh-X8XpF4sj9t/agent.20083

```

```

CPATH=/opt/intel/compilers_and_libraries_2019.3.199/
↪ linux/ipp/include:/opt/intel/compilers_and_libra
↪ ries_2019.3.199/linux/mkl/include:/opt/intel/com
↪ pilers_and_libraries_2019.3.199/linux/pstl/inclu
↪ de:/opt/intel/compilers_and_libraries_2019.3.199
↪ /linux/tbb/include:/opt/intel/compilers_and_libr
↪ aries_2019.3.199/linux/daal/include
TMOUT=604800
NLSPATH=/opt/intel/compilers_and_libraries_2019.3.19
↪ 9/linux/compiler/lib/intel64/locale/%l_%t/%N:/op
↪ t/intel/compilers_and_libraries_2019.3.199/linux
↪ /mkl/lib/intel64_lin/locale/%l_%t/%N:/opt/intel/
↪ debugger_2019/gdb/intel64/share/locale/%l_%t/%N
PATH=/opt/intel/advisor_2019.3.0.591490/bin64:/opt/i
↪ ntel/vtune_amplifier_2019.3.0.591499/bin64:/opt/
↪ intel/inspector_2019.3.0.591484/bin64:/opt/intel
↪ /itac/2019.3.032/intel64/bin:/opt/intel/clck/201
↪ 9.2.1/bin/intel64:/opt/intel/compilers_and_libra
↪ ries_2019.3.199/linux/bin/intel64:/opt/intel/com
↪ pilers_and_libraries_2019.3.199/linux/mpi/intel6
↪ 4/libfabric/bin:/opt/intel/compilers_and_librari
↪ es_2019.3.199/linux/mpi/intel64/bin:/opt/intel/d
↪ ebugger_2019/gdb/intel64/bin:/opt/packages/pgi/l
↪ inux86-64/19.4/mpi/openmpi-3.1.3/bin:/opt/packag
↪ es/pgi/linux86-64-llvm/19.4/bin:/usr/lib64/qt-3.
↪ 3/bin:/opt/packages/xdusage/2.1-1/bin:/opt/packa
↪ ges/slurm/default/bin:/opt/packages/allocations:
↪ /opt/packages/interact/bin:/usr/lib64/ccache:/us
↪ r/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/
↪ opt/puppetlabs/puppet/bin:/bin:/sbin:/opt/puppet
↪ labs/bin:/:/home/USER/.local/bin:/home/USER/bin
↪ :/opt/intel/parallel_studio_xe_2019.3.062/bin
MAIL=/var/spool/mail/USER
VT_ADD_LIBS=-ldwarf -lelf -ltunwind -lm -lpthread
_=/usr/bin/env
TBBROOT=/opt/intel/compilers_and_libraries_2019.3.19
↪ 9/linux/tbb
PWD=/home/USER/distributeddynamicprogramming/DMCode_
↪ decoupled/Floyd_Warshall_APSP
MYPROXY_SERVER_PORT=7512
_LMFILES=/opt/modulefiles/psc_path/1.1:/opt/modulef
↪ iles/slurm/default:/opt/modulefiles/xdusage/2.1-
↪ 1:/opt/modulefiles/pgi/19.4:/opt/modulefiles/mpi
↪ /pgi_openmpi/19.4:/opt/modulefiles/intel/19.3
LANG=en_US.UTF-8
MODULEPATH=/opt/modulefiles
LOADEDMODULES=psc_path/1.1:slurm/default:xdusage/2.1
↪ -1:pgi/19.4:mpi/pgi_openmpi/19.4:intel/19.3
MYPROXY_SERVER=myproxy.xsede.org,myproxy.psc.xsede.o
↪ rg
LM_LICENSE_FILE=/opt/packages/pgi/license.dat
DAALROOT=/opt/intel/compilers_and_libraries_2019.3.1
↪ 99/linux/daal
KRB5CCNAME=FILE:/tmp/krb5cc-57976_5wQRFx6STv
HISTCONTROL=ignoredups
HOME=/home/USER

```

D2P: From Recursive Formulations to Distributed-Memory Codes

```

SHLVL=2
GLOBUS_TCP_PORT_RANGE=50000,51000
VT_ARCH=intel64
LOGNAME=USER
CVS_RSH=ssh
GLOBUS_TCP_SOURCE_RANGE=50000,51000
QTLIB=/usr/lib64/qt-3.3/lib
SSH_CONNECTION=128.46.4.86 33678 128.182.108.57 22
CLASSPATH=/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib/mpi.jar:/opt/intel/compilers_and_libraries_2019.3.199/linux/daal/lib/daal.jar
MODULESHOME=/usr/share/Modules
LESSOPEN=||/usr/bin/lesspipe.sh %s
PKG_CONFIG_PATH=/opt/intel/advisor_2019.3.0.591490/include/pkgconfig/lib64:/opt/intel/compilers_and_libraries_2019.3.199/linux/mkl/bin/pkgconfig
INFOPATH=/opt/intel/documentation_2019/en/debugger/gdb-ia/info/
DISPLAY=10.4.212.75:27.0
XDG_RUNTIME_DIR=/run/user/57976
SCRATCH=/pylon5/ac560jp/USER
I_MPI_ROOT=/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi
BASH_FUNC_module()=( { eval ` /usr/bin/modulecmd bash $* `
}
++ lsb_release -a
LSB Version:           :core-4.1-amd64:core-4.1-noarch:desktop-4.1-amd64:desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:printing-4.1-amd64:printing-4.1-noarch
Distributor ID:        CentOS
Description:            CentOS Linux release 7.4.1708 (Core)
Release:                7.4.1708
Codename:               Core
++ uname -a
Linux br006.pvt.bridges.psc.edu 3.10.0-693.21.1.el7.x86_64 #1 SMP Wed Mar 7 19:03:37 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
++ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                28
On-line CPU(s) list:   0-27
Thread(s) per core:    1
Core(s) per socket:    14
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                63

```

```

Model name:            Intel(R) Xeon(R) CPU E5-2695 v3
    @ 2.30GHz
Stepping:              2
CPU MHz:               2300.000
CPU max MHz:           2300.0000
CPU min MHz:           1200.0000
BogoMIPS:              4594.41
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              35840K
NUMA node0 CPU(s):    0-6,14-20
NUMA node1 CPU(s):    7-13,21-27
Flags:                 fpu vme de pse tsc msr pae mce
    cx8 apic sep mtrr pge mca cmov pat pse36 clflush
    dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
    pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs
    bts rep_good nopl xtopology nonstop_tsc
    aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
    ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm
    pcid dca sse4_1 sse4_2 x2apic movbe popcnt
    tsc_deadline_timer aes xsave avx f16c rdrand
    lahf_lm abm epb invpcid_single spec_ctrl
    ibpb_support tpr_shadow vnmi flexpriority ept
    vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
    invpcid cqm xsaveopt cqm_llc cqm_occup_llc dtherm
    ida arat pln pts
++ cat /proc/meminfo
MemTotal:              131734432 kB
MemFree:               41752760 kB
MemAvailable:          108653876 kB
Buffers:               2867052 kB
Cached:                54055392 kB
SwapCached:            22208 kB
Active:                20943252 kB
Inactive:              40462636 kB
Active(anon):          4893564 kB
Inactive(anon):        5441100 kB
Active(file):          16049688 kB
Inactive(file):        35021536 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             17591292 kB
SwapFree:              17414260 kB
Dirty:                 20 kB
Writeback:             0 kB
AnonPages:             4463276 kB
Mapped:                243280 kB
Shmem:                 5851192 kB
Slab:                  26899712 kB
SReclaimable:          17144064 kB
SUnreclaim:            9755648 kB
KernelStack:           17968 kB
PageTables:            104808 kB
NFS_Unstable:          0 kB

```

```

Bounce:          0 kB
WritebackTmp:    0 kB
CommitLimit:     83458508 kB
Committed_AS:    14595576 kB
VmallocTotal:    34359738367 kB
VmallocUsed:      766628 kB
VmallocChunk:    34291796908 kB
HardwareCorrupted: 0 kB
AnonHugePages:   1241088 kB
HugePages_Total: 0
HugePages_Free:  0
HugePages_Rsvd:  0
HugePages_Surp:  0
Hugepagesize:    2048 kB
DirectMap4k:     637696 kB
DirectMap2M:     65294336 kB
DirectMap1G:     70254592 kB
++ inxi -F -c0
./collect_environment.sh: line 7: inxi: command not
found
++ lsblk -a
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda   8:0    0   1G  0 disk
sdb   8:16   0  2.7T  0 disk
├─sdb1 8:17   0    1M  0 part
└─sdb2 8:18   0 300G  0 part /
sdc   8:32   0  3.7T  0 disk
sdd   8:48   0  3.7T  0 disk
sde   8:64   0  3.7T  0 disk
loop0 7:0     0      1 loop
loop1 7:1     0      1 loop
loop2 7:2     0    4G  1 loop
++ lsscsi -s
[0:0:0:0] disk HP iLO LUN 00 Media 0 2.10
└─ /dev/sda 1.07GB
[1:0:0:0] disk ATA MB3000GCWDB HPGE
└─ /dev/sdb 3.00TB
[2:0:0:0] disk ATA HGST HDN724040AL A5E0
└─ /dev/sdc 4.00TB
[3:0:0:0] disk ATA HGST HDN724040AL A5E0
└─ /dev/sdd 4.00TB
[4:0:0:0] disk ATA HGST HDN724040AL A5E0
└─ /dev/sde 4.00TB
++ module list
+++ /usr/bin/modulecmd bash list
Currently Loaded Modulefiles:
  1) psc_path/1.1          2) slurm/default
  3) xdusage/2.1-1        4) pgi/19.4
  5) mpi/pgi_openmpi/19.4 6) intel/19.3
++ eval
++ nvidia-smi
./collect_environment.sh: line 11: nvidia-smi:
command not found
++ lshw -short -quiet -sanitize
++ cat

```

```

./collect_environment.sh: line 12: lshw: command not
found
++ lspci
00:00.0 Host bridge: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMI2 (rev 02)
00:01.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 1 (rev 02)
00:01.1 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 1 (rev 02)
00:02.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)
00:02.1 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)
00:02.2 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)
00:02.3 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)
00:03.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)
00:03.1 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)
00:03.2 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)
00:03.3 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
└─ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)
00:04.0 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 0 (rev 02)
00:04.1 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 1 (rev 02)
00:04.2 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 2 (rev 02)
00:04.3 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 3 (rev 02)
00:04.4 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 4 (rev 02)
00:04.5 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 5 (rev 02)
00:04.6 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 6 (rev 02)
00:04.7 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 DMA Channel 7 (rev 02)
00:05.0 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 Address Map, VTd_Misc,
  System Management (rev 02)
00:05.1 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 Hot Plug (rev 02)
00:05.2 System peripheral: Intel Corporation Xeon E7
└─ v3/Xeon E5 v3/Core i7 RAS, Control Status and
  Global Errors (rev 02)
00:05.4 PIC: Intel Corporation Xeon E7 v3/Xeon E5
└─ v3/Core i7 I/O APIC (rev 02)
00:11.0 Unassigned class [ff00]: Intel Corporation
└─ C610/X99 series chipset SPSR (rev 05)

```

D2P: From Recursive Formulations to Distributed-Memory Codes

00:14.0 USB controller: Intel Corporation C610/X99
↳ series chipset USB xHCI Host Controller (rev 05)
00:1a.0 USB controller: Intel Corporation C610/X99
↳ series chipset USB Enhanced Host Controller #2
↳ (rev 05)
00:1c.0 PCI bridge: Intel Corporation C610/X99 series
↳ chipset PCI Express Root Port #1 (rev d5)
00:1c.2 PCI bridge: Intel Corporation C610/X99 series
↳ chipset PCI Express Root Port #3 (rev d5)
00:1c.4 PCI bridge: Intel Corporation C610/X99 series
↳ chipset PCI Express Root Port #5 (rev d5)
00:1c.6 PCI bridge: Intel Corporation C610/X99 series
↳ chipset PCI Express Root Port #7 (rev d5)
00:1c.7 PCI bridge: Intel Corporation C610/X99 series
↳ chipset PCI Express Root Port #8 (rev d5)
00:1d.0 USB controller: Intel Corporation C610/X99
↳ series chipset USB Enhanced Host Controller #1
↳ (rev 05)
00:1f.0 ISA bridge: Intel Corporation C610/X99 series
↳ chipset LPC Controller (rev 05)
00:1f.2 SATA controller: Intel Corporation C610/X99
↳ series chipset 6-Port SATA Controller [AHCI mode]
↳ (rev 05)
00:1f.3 SMBus: Intel Corporation C610/X99 series
↳ chipset SMBus Controller (rev 05)
01:00.0 System peripheral: Hewlett-Packard Company
↳ Integrated Lights-Out Standard Slave
↳ Instrumentation & System Support (rev 06)
01:00.1 VGA compatible controller: Matrox Electronics
↳ Systems Ltd. MGA G200EH (rev 01)
01:00.2 System peripheral: Hewlett-Packard Company
↳ Integrated Lights-Out Standard Management
↳ Processor Support and Messaging (rev 06)
01:00.4 USB controller: Hewlett-Packard Company
↳ Integrated Lights-Out Standard Virtual USB
↳ Controller (rev 03)
02:00.0 Ethernet controller: Broadcom Limited
↳ NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
02:00.1 Ethernet controller: Broadcom Limited
↳ NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
02:00.2 Ethernet controller: Broadcom Limited
↳ NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
02:00.3 Ethernet controller: Broadcom Limited
↳ NetXtreme BCM5719 Gigabit Ethernet PCIe (rev 01)
08:00.0 Fabric controller: Intel Corporation Omni-Path
↳ HFI Silicon 100 Series [discrete] (rev 10)
7f:08.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 0 (rev 02)
7f:08.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 0 (rev 02)
7f:09.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 1 (rev 02)
7f:09.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 1 (rev 02)
7f:0b.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 R3 QPI Link 0 & 1 Monitoring
↳ (rev 02)
7f:0b.1 Performance counters: Intel Corporation Xeon
↳ E7 v3/Xeon E5 v3/Core i7 R3 QPI Link 0 & 1
↳ Monitoring (rev 02)
7f:0b.2 Performance counters: Intel Corporation Xeon
↳ E7 v3/Xeon E5 v3/Core i7 R3 QPI Link 0 & 1
↳ Monitoring (rev 02)
7f:0c.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.5 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.6 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0c.7 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0d.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0d.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0d.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0d.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0d.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0d.5 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)
7f:0f.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)
7f:0f.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)
7f:0f.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)
7f:0f.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)
7f:0f.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 System Address Decoder &
↳ Broadcast Registers (rev 02)
7f:0f.5 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 System Address Decoder &
↳ Broadcast Registers (rev 02)
7f:0f.6 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 System Address Decoder &
↳ Broadcast Registers (rev 02)

7f:10.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 PCIe Ring Interface (rev 02)

7f:10.1 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 PCIe Ring Interface (rev 02)

7f:10.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Scratchpad & Semaphore
 ↳ Registers (rev 02)

7f:10.6 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 Scratchpad & Semaphore
 ↳ Registers (rev 02)

7f:10.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Scratchpad & Semaphore
 ↳ Registers (rev 02)

7f:12.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 0 (rev 02)

7f:12.1 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 Home Agent 0 (rev 02)

7f:12.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 0 Debug (rev 02)

7f:12.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 1 (rev 02)

7f:12.5 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 Home Agent 1 (rev 02)

7f:12.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 1 Debug (rev 02)

7f:13.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Target Address, Thermal & RAS Registers (rev 02)

7f:13.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Target Address, Thermal & RAS Registers (rev 02)

7f:13.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Channel Target Address Decoder (rev 02)

7f:13.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Channel Target Address Decoder (rev 02)

7f:13.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Channel 0/1 Broadcast
 ↳ (rev 02)

7f:13.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Global Broadcast (rev 02)

7f:14.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 0 Thermal Control (rev 02)

7f:14.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 1 Thermal Control (rev 02)

7f:14.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 0 ERROR Registers (rev 02)

7f:14.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 1 ERROR Registers (rev 02)

7f:14.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

7f:14.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

7f:14.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

7f:14.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

7f:16.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Target Address, Thermal & RAS Registers (rev 02)

7f:16.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Target Address, Thermal & RAS Registers (rev 02)

7f:16.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Channel Target Address Decoder (rev 02)

7f:16.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Channel Target Address Decoder (rev 02)

7f:16.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Channel 2/3 Broadcast
 ↳ (rev 02)

7f:16.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Global Broadcast (rev 02)

7f:17.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 0 Thermal Control (rev 02)

7f:17.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 1 Thermal Control (rev 02)

7f:17.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 0 ERROR Registers (rev 02)

7f:17.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 1 ERROR Registers (rev 02)

7f:17.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

7f:17.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

7f:17.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

7f:17.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

7f:1e.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)

7f:1e.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)

7f:1e.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)

7f:1e.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)

D2P: From Recursive Formulations to Distributed-Memory Codes

7f:1e.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)

7f:1f.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 VCU (rev 02)

7f:1f.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 VCU (rev 02)

80:00.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 0 (rev 02)

80:01.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 1 (rev 02)

80:01.1 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 1 (rev 02)

80:02.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)

80:02.1 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)

80:02.2 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)

80:02.3 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 2 (rev 02)

80:03.0 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)

80:03.1 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)

80:03.2 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)

80:03.3 PCI bridge: Intel Corporation Xeon E7 v3/Xeon
↳ E5 v3/Core i7 PCI Express Root Port 3 (rev 02)

80:04.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 0 (rev 02)

80:04.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 1 (rev 02)

80:04.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 2 (rev 02)

80:04.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 3 (rev 02)

80:04.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 4 (rev 02)

80:04.5 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 5 (rev 02)

80:04.6 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 6 (rev 02)

80:04.7 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 DMA Channel 7 (rev 02)

80:05.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Address Map, VTd_Misc,
↳ System Management (rev 02)

80:05.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Hot Plug (rev 02)

80:05.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 RAS, Control Status and
↳ Global Errors (rev 02)

80:05.4 PIC: Intel Corporation Xeon E7 v3/Xeon E5
↳ v3/Core i7 I/O APIC (rev 02)

ff:08.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 0 (rev 02)

ff:08.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 0 (rev 02)

ff:09.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 1 (rev 02)

ff:09.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 QPI Link 1 (rev 02)

ff:0b.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 R3 QPI Link 0 & 1 Monitoring
↳ (rev 02)

ff:0b.1 Performance counters: Intel Corporation Xeon
↳ E7 v3/Xeon E5 v3/Core i7 R3 QPI Link 0 & 1
↳ Monitoring (rev 02)

ff:0b.2 Performance counters: Intel Corporation Xeon
↳ E7 v3/Xeon E5 v3/Core i7 R3 QPI Link 0 & 1
↳ Monitoring (rev 02)

ff:0c.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.5 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.6 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0c.7 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0d.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0d.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0d.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0d.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0d.4 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0d.5 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Unicast Registers (rev 02)

ff:0f.0 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)

ff:0f.1 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)

ff:0f.2 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)

ff:0f.3 System peripheral: Intel Corporation Xeon E7
↳ v3/Xeon E5 v3/Core i7 Buffered Ring Agent (rev 02)

ff:0f.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 System Address Decoder &
 ↳ Broadcast Registers (rev 02)

ff:0f.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 System Address Decoder &
 ↳ Broadcast Registers (rev 02)

ff:0f.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 System Address Decoder &
 ↳ Broadcast Registers (rev 02)

ff:10.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 PCIe Ring Interface (rev 02)

ff:10.1 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 PCIe Ring Interface (rev
 ↳ 02)

ff:10.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Scratchpad & Semaphore
 ↳ Registers (rev 02)

ff:10.6 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 Scratchpad & Semaphore
 ↳ Registers (rev 02)

ff:10.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Scratchpad & Semaphore
 ↳ Registers (rev 02)

ff:12.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 0 (rev 02)

ff:12.1 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 Home Agent 0 (rev 02)

ff:12.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 0 Debug (rev 02)

ff:12.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 1 (rev 02)

ff:12.5 Performance counters: Intel Corporation Xeon
 ↳ E7 v3/Xeon E5 v3/Core i7 Home Agent 1 (rev 02)

ff:12.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Home Agent 1 Debug (rev 02)

ff:13.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Target Address, Thermal & RAS Registers (rev 02)

ff:13.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Target Address, Thermal & RAS Registers (rev 02)

ff:13.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Channel Target Address Decoder (rev 02)

ff:13.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 0 Channel Target Address Decoder (rev 02)

ff:13.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Channel 0/1 Broadcast
 ↳ (rev 02)

ff:13.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Global Broadcast (rev
 ↳ 02)

ff:14.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 0 Thermal Control (rev 02)

ff:14.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 1 Thermal Control (rev 02)

ff:14.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 0 ERROR Registers (rev 02)

ff:14.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 0 Channel 1 ERROR Registers (rev 02)

ff:14.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

ff:14.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

ff:14.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

ff:14.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 0 & 1 (rev 02)

ff:16.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Target Address, Thermal & RAS Registers (rev 02)

ff:16.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Target Address, Thermal & RAS Registers (rev 02)

ff:16.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Channel Target Address Decoder (rev 02)

ff:16.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory Controller
 ↳ 1 Channel Target Address Decoder (rev 02)

ff:16.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Channel 2/3 Broadcast
 ↳ (rev 02)

ff:16.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO Global Broadcast (rev
 ↳ 02)

ff:17.0 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 0 Thermal Control (rev 02)

ff:17.1 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 1 Thermal Control (rev 02)

ff:17.2 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 0 ERROR Registers (rev 02)

ff:17.3 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 Integrated Memory
 ↳ Controller 1 Channel 1 ERROR Registers (rev 02)

ff:17.4 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

ff:17.5 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

ff:17.6 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

ff:17.7 System peripheral: Intel Corporation Xeon E7
 ↳ v3/Xeon E5 v3/Core i7 DDRIO (VMSE) 2 & 3 (rev 02)

```

ff:1e.0 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)
ff:1e.1 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)
ff:1e.2 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)
ff:1e.3 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)
ff:1e.4 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 Power Control Unit (rev 02)
ff:1f.0 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 VCU (rev 02)
ff:1f.2 System peripheral: Intel Corporation Xeon E7
↪ v3/Xeon E5 v3/Core i7 VCU (rev 02)

```

ARTIFACT EVALUATION

Verification and validation studies: Validating Smith-Waterman implementation: The optimal score computed is compared with that obtained from a publicly available hand-written software (<https://www.alexjf.net/projects/distributed-systems/smith-waterman-openmp-and-openmpi/>) and found that the results matched. Validating Matrix Chain Multiplication (MCM) implementation: For checking the correctness of results, we provided the input values used in the example mentioned in CLRS book (Introduction to Algorithms, Third edition, Figure 15.5) and observed that the D2P output matched. Validating All Pairs Shortest Path implementation : we saw that the shortest path from a randomly source to destination vertex matched manually computed value for a toy graph. In addition to these, all the benchmarks except the RNA problem and Floyd_Warshall_APSP are validated against the standard iterative implementations written in-house. We could not do an validation of the Minimum Weight Triangulation (MWT) and the RNA problem (Nussinov4D) by comparing with reference software because of non-availability of such software.

Accuracy and precision of timings: The runtimes are measured until a steady state is achieved. i.e. $\pm 6\%$ of the mean runtime with 95% confidence. In one experiment (D2P vs. UPC) where the runtime was very small, the margin of error was very small 0.08% with 95% confidence.