COTS Clusters vs. the Earth Simulator:
An Application Study Using IMPACT-3D

Daniel Chavarría-Miranda Guohua Jin John Mellor-Crummey
Dept. of Computer Science
Rice University
{danich,jin,johnmc}@cs.rice.edu

Abstract

In 2002, Japan announced the Earth Simulator—a supercomputer based on low-volume vector processors and a custom network—and reported that computational scientists had used it to achieve 14.9 TFLOPS with the IMPACT-3D code, which is written in High Performance Fortran (HPF). Of particular interest was that they had achieved this level of performance using a high-level parallel programming model. There has been considerable concern in the U.S. about the appropriateness of its hardware and software investments in supercomputing technology. To help assess the U.S. strategy of building systems from commodity-off-the-shelf (COTS) components, we explored using a combination of HPF and scalar compiler technology to tailor IMPACT-3D to microprocessor-based supercomputers and evaluated its performance and scalability on the AlphaServer-based Lemieux cluster at the Pittsburgh Supercomputer Center (PSC). On the Earth Simulator, IMPACT-3D achieved 45% of peak performance on 4096 processors; on 1024 processors of PSC’s Lemieux, we achieved 17.29% of peak performance.

1. Introduction

In April 2002, Japan stunned the U.S. supercomputing community when it unveiled the Earth Simulator [18] with a peak performance of 40 TFLOPS [6]. This caused considerable concern within the supercomputing community in the U.S. since, at the time it was announced, the Earth Simulator matched the raw performance of the top 20 American supercomputers combined [11].

The design of the Earth Simulator, which is based on vector processors and an extremely high bandwidth crossbar interconnect, differs significantly from that of the microprocessor-based systems that have been primary focus of investment in the U.S. What has been most striking about the Earth Simulator has been its ability to deliver high performance on complex scientific applications. In 2002, Earth Simulator researchers garnered three Gordon Bell awards at SC2002 for applications that achieved 14.9–28.58 TFLOPS [17, 20, 21].

Since the Earth Simulator’s introduction, there has been considerable interest in comparing its capabilities to that of microprocessor-based systems. A study by Kerbyson et al. [10] used performance models of ASCI applications to project performance on the Earth Simulator and compare it with the performance of the four top DOE microprocessor-based supercomputers. Recently, Oliker et al. [13] compared the Earth Simulator with SGI Altix, Cray X1, as well as IBM Power3 and Power4 systems using four science applications: LBMHD (a magneto-hydrodynamics application using the Lattice Botzmann method), PARATEC [14] (a materials science code that solves the Kohn-Sham equations), Cactus (an astrophysics code that solves Einstein’s equations), and GTC (a magnetic fusion particle-in-cell code that solves the Vlasov-Poisson equations). With these applications, they reported achieving 4–7% of peak performance using 1024 processors of a Power3-based system (the largest system they measured) for all but PARATEC, which achieved 28% efficiency on 512 processors (the largest configuration measured for this application). In contrast, the Earth Simulator achieved efficiencies ranging from 34-41% on 1024 processors for all but GTC, which achieved an efficiency of 16%. Their experiments clearly showed that in practice the Earth Simulator achieves much higher efficiencies than microprocessor-based systems.

One particularly significant achievement on the Earth
Simulator was that IMPACT-3D, a plasma simulation application written in High-Performance Fortran (HPF), achieved 45% of the machine’s peak performance [17]. In part, this result rekindled interest within the U.S. in high-level data programming models for parallel systems. In the prior comparison studies, one question that has not been answered is how well the performance of microprocessor-based systems can compare with the Earth Simulator for high-level data-parallel programs. In particular, can compilers for high-level data-parallel languages adequately tailor applications to achieve respectable efficiency on large-scale microprocessor-based clusters? In this paper, we address this question with a study of IMPACT-3D on the Pittsburgh Supercomputer Center’s Lemieux AlphaServer/SC cluster.

Microprocessor-based supercomputers are typically based on commodity “blade”-style compute nodes, consisting of a small number of processors in an SMP configuration, linked by interconnects ranging from commodity Ethernet to specialized fabrics such as InfiniBand, Quadrics or Myrinet [3, 8, 15]. Microprocessor-based clusters can only achieve high performance when an application’s data accesses exhibit good locality and exploit caches effectively. Memory bandwidth on such systems is only a fraction of that available on the Earth Simulator.

To parallelize and tailor variants of IMPACT-3D for microprocessor-based clusters, we used two tools: dHPF, Rice University’s research compiler for HPF [1, 2, 5, 12], and LoopTool, a source-to-source tool for optimizing memory hierarchy utilization [16]. The dHPF compiler employs sophisticated analyses and code generation techniques to produce highly-tuned and flexible SPMD applications from implicitly-parallel HPF sources.

Our results show that a program written in a high-level parallel programming model can be compiled into code that is efficient on large-scale microprocessor-based systems. Using the dHPF compiler to parallelize IMPACT-3D, we achieved 17.29% of the peak performance (354.17 GFLOPS out of 2048 GFLOPS) of a 1024-processor partition of the Lemieux cluster at the Pittsburgh Supercomputing Center. It is notable that the performance we were able to achieve with a compiler-based parallelization of an HPF program on this cluster is competitive with the performance of hand-coded parallelizations of other scientific codes on large-scale microprocessor-based clusters [13].

Section 2 provides background information about the dHPF compiler and the IMPACT-3D application. Section 3 describes how we used dHPF and LoopTool to parallelize and optimize IMPACT-3D for microprocessor-based clusters. This section also explores the impact of several manual optimizations that could be incorporated into our source-level compilation tools. Section 4 discusses several challenges we face in generating better parallel codes for IMPACT-3D on microprocessor-based clusters. Section 5 presents our conclusions.

2. Background

The dHPF Compiler. dHPF is a research compiler that has been under development at Rice University over the past 8 years. The dHPF project has focused on design and experimental evaluation of sophisticated analyses and code generation techniques for High-Performance Fortran. dHPF transforms an HPF program into a single-program-multiple data (SPMD) Fortran program that uses a runtime library built upon MPI to perform interprocess communication. dHPF principally supports whole-program compilation of HPF programs written in a Fortran 77 style, although it also supports key Fortran 90 language constructs such as array sections, interface blocks and dynamically-allocated arrays. dHPF supports a broad spectrum of optimizations including a rich set of communication optimizations (e.g. vectorization, coalescing, and aggregation), partial replication of computation to reduce communication, and support for analysis and code generation in the presence of symbolic quantities (e.g. the number of processors). In addition, dHPF supports extended directives for controlling data movement that were inspired by HPF/JA [19], the Japanese HPF project. The dHPF compiler is described in more detail elsewhere [1, 2, 5, 4, 12].

IMPACT-3D. The IMPACT-3D [17] (IMPlosion Analysis Code with TVD [7] scheme) application simulates a three-dimensional Rayleigh-Taylor instability in a spherical system as part of a larger plasma fluid dynamics simulation. The simulation requires very high resolution to compute scientifically relevant results and for this reason must be executed at a very large scale.

IMPACT-3D was developed for the Earth Simulator and was written in HPF/ES, a dialect of HPF supported only on that system, which includes support for the HPF/JA extensions [19]. The application is structured as six subroutines and a main program. Three routines perform the bulk of the computation: adv3dx, adv3dy, and adv3dz. These routines perform the computation for one step of the simulation along each of the three spatial dimensions. The main program contains a time step loop. There are six main distributed arrays and twenty-three auxiliary distributed arrays. All twenty-nine arrays are passed as parameters to each of the adv3d [xyz] routines (we use the shorthand form adv3d [xyz] to refer to the routines adv3dx, adv3dy and adv3dz).

The HPF implementation of IMPACT-3D for the Earth Simulator used a uniform 1D BLOCK distribution of the twenty-nine arrays in the third (z) dimension; this distribution enables each processor to execute
the adv3dx and adv3dy routines independently without any communication. In contrast, adv3dz requires four phases of loosely-synchronous communication between parallel three-dimensional loop nests. HPF/JA REFLECT and LOCAL directives are used to control placement of each loosely-synchronous communication.

3. IMPACT-3D for Clusters

Microprocessor-based supercomputers lack the enormous memory and communication bandwidth of the Earth Simulator. Tailoring scientific applications to a microprocessor-based cluster requires exploiting cache locality to compensate for the lack of memory bandwidth and reducing the applications’ dependence on high-bandwidth, low latency communication.

To evaluate the performance of IMPACT-3D on a microprocessor-based cluster, we compiled the original source (as written for the Earth Simulator) with dHPF, and executed it on the Lemieux Alpha cluster at the Pittsburgh Supercomputing Center. Lemieux consists of 750 1GHz 4-processor nodes, each having 4 GB of RAM, a 64KB 2-way set associative primary cache and an 8MB 2-way set associative secondary cache. Latencies to the primary cache, secondary cache, and memory for floating point loads are 4, 12, and 80 cycles respectively. Each processor has a peak capacity of 2 GFLOPS. The cluster uses a Quadrics ELAN3 interconnection network, and runs HP’s Tru64 UNIX version. All codes were compiled using HP’s native compilers for this platform (with the -O5 -fast flags), and linked against a Quadrics-aware MPI library. The per-node memory limits the problem sizes we can execute on the cluster, and the partition choices limit the number of processors on which the application can be executed.

All of our experiments were conducted using IMPACT-3D’s 1024 × 1024 × 2048 problem size—one of the problem sizes studied on the Earth Simulator. Due to insufficient memory, we were unable to study the 2048 × 2048 × 4096 problem size with which the best Earth Simulator results were achieved [17].

Although dHPF also supports the HPF/JA directives used in the original IMPACT-3D source code, in most cases they were unnecessary for dHPF; dHPF is able to automatically generate the loosely-synchronous communication needed by analyzing the application’s data access patterns. However, to avoid redundantly communicating boundary values for several arrays (sr, sm, sp, se, sn and s1), dHPF needs an HPF/JA REFLECT directive to explicitly fill the shadow regions for these arrays once and then annotated the loops in which they were used with an HPF/JA LOCAL directive to indicate that no additional communication is needed for these arrays.

Table 1 presents the results for the experiments conducted using the original code. The first column of the table lists the number of processors used, the second column lists the relative speedup of the code when compared to the 128-processor execution, the third column lists GFLOPS 1, the fourth column lists the fraction of the peak floating-point performance achieved.

The dHPF compiler applied several automatic optimizations to this code, including communication vectorization and aggregation, as well as loop & array bounds reduction. However, even with these optimizations the dHPF-generated code delivers only a small fraction of peak performance. Also, the performance of the dHPF-generated code for this 1D BLOCK partitioning does not scale linearly as the number of processors increases. These results show that the original code and its partitioning, though well suited for the vector-based, high-bandwidth model of the Earth Simulator, is not ideal for microprocessor-based clusters. Our analysis of the code revealed two principal reasons for the low performance and lack of scalability.

- The surface-to-volume ratio of the 1D data partition increases linearly with the number of processors. A large volume of data must be communicated in the adv3dz routine.
- The use of large temporary arrays leads to a large working set for each processor, which can degrade scalar performance. We discuss this in more detail in Section 3.2.

3.1. Data Partitioning

The 1D partition used in the original IMPACT-3D code, though efficient with the high-bandwidth interconnect of the Earth Simulator, can be problematic for clusters with more modest interconnects. For example, consider the 1024 ×

<table>
<thead>
<tr>
<th># of procs</th>
<th>Rel. Speedup</th>
<th>GFLOPS</th>
<th>% peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.00</td>
<td>35.49</td>
<td>13.86%</td>
</tr>
<tr>
<td>256</td>
<td>1.87</td>
<td>66.34</td>
<td>12.96%</td>
</tr>
<tr>
<td>512</td>
<td>2.80</td>
<td>99.27</td>
<td>9.69%</td>
</tr>
<tr>
<td>1024</td>
<td>4.02</td>
<td>142.66</td>
<td>6.97%</td>
</tr>
</tbody>
</table>

1 GFLOPS were computed using hardware counters on a processor of Lemieux to count floating point operations in a sequential execution of IMPACT-3D using the data size corresponding to one processor’s data in a 512 processor execution (1024 × 1024 × 4). This number was then scaled appropriately for each number of processors (128, 256, 512 and 1024) and then divided by the total time spent in the application (calculated with mpi_wtime()).
1024 × 2048 problem size studied by Sakagami et al. [17] on the Earth Simulator. If we partition this problem along the z dimension on 1024 processors, then the local size of an array on one processor is 1024 × 1024 × 2, with each processor holding two complete z planes. The communication phases in adv3dz require sending a complete plane of several arrays between processors. The ratio of communicated data to the data volume owned by a processor (for one array) is 50%. Each plane consists of 10242 single precision floating point numbers (4 MB of data). The communication volume required using a 1D partitioning is quite high.

A higher-dimensional data partitioning can be used to reduce the communication volume. When using a 3D data partitioning, communication is required in adv3dx and adv3dy as well as in adv3dz. With a 3D partitioning, each routine requires four loosely-synchronous communication phases between parallel loop nests. A 3D partitioning reduces the total communication bandwidth required, though it requires three times as many communication events as a 1D partitioning. A 3D BLOCK distribution of the 1024 × 1024 × 2048 problem onto 1024 processors (viewed as a 8 × 8 × 16 mesh) yields a compact local array section of 128 × 128 × 128 elements on each processor, in contrast to the thin 1024 × 1024 × 2 slab obtained when using a 1D distribution. The amount of communication required depends on the surface to volume ratio of the partitioned data. The surface to volume ratio of the 3D partitioning is much lower than that of the 1D partitioning. With the 3D partitioning, each communication phase exchanges one plane of the local array section, namely 1282 elements. In the 1D distribution, the size of the plane exchanged is 10242 elements. Even though, the 3D partitioning requires three times as many communication phases as the 1D partitioning, its aggregate communication volume is only 4.69% of that of the 1D distribution.

### 3.1.1. IMPACT-3D with 3D Distributions

As mentioned in Section 2, the bulk of the computation in IMPACT-3D occurs in the adv3d[xyz] routines. In the original HPF/ES implementation of the application, the only routine that requires communication is adv3dz. This routine has four loosely-synchronous communication phases interleaved with five fully parallel 3D loops. Table 2 shows this structure in more detail, each data movement phase or loop shows the names of the distributed arrays that it operates upon. Each communication phase shows the direction of data movement along the principal array dimension operated upon by the procedure. If the principal array dimension is partitioned, communication is required with the processors that have data to the left or right along the principal dimension.

As an alternative to the 1D (*, *, BLOCK) data distribution of the original source code, here we consider the performance of using a 3D (BLOCK, BLOCK, BLOCK) data distribution in which all data dimensions are partitioned. The 1D block partitioning distributes the arrays onto a 1D processor array with a symbolic number of processors (number_of_processors()). The 3D partition strategy distributes the arrays onto a power-of-two number of processors organized as a compact rectangular mesh, i.e. 512 processors are organized as a 8 × 8 × 8 mesh and 1024 processors are organized as an 8 × 8 × 16 mesh.

Table 3 presents the results for experiments conducted using a 3D BLOCK partitioning. These experiments were performed using an HPF code that is essentially the same as the original sources, except for the change to the data distribution and the addition of REFLECT and LOCAL directives to control the new communication adv3dx and adv3dy. The new REFLECT and LOCAL directives in adv3dx and adv3dy are analogous to those already present in adv3dz. With the 3D partitioning, we observe a performance improvement of 146% over the 1D version on 1024 processors.

### 3.2. Improving Node Performance

In IMPACT-3D, the 3D TVD scheme is performed along each axial direction during each time step. Complex calculations are factored into multiple steps, with each step performing relative simple computation to advance some aspect of the program state. As a result, individual loop nests consume values of one or more arrays, and compute results into new arrays that will be used by subsequent loop nests. Although the original IMPACT-3D code replicated some computation to avoid creating some full-size three-dimensional array temporaries the code still used 29 3D arrays in each adv3d[xyz] routine. For large problem sizes, these arrays have a large memory footprint. Using so many large arrays leads to low cache performance because values brought in to cache in one loop nest are evicted from cache before they can be reused in another. Efficient cache utilization is important for microprocessor-based systems since they have much lower processor-to-memory bandwidth than vector machines such as the Earth Simulator.

To improve cache utilization and boost node performance, we transformed the routines adv3dx and adv3dy.
Table 2. Pseudocode for adv3d[xyz].

<table>
<thead>
<tr>
<th>Phase</th>
<th>Data Transferred or Computed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st comm. phase</td>
<td>sr, sm, sp, se, sn, sl (shift left)</td>
</tr>
<tr>
<td>1st and 2nd loop nests</td>
<td>compute walfa1-walfa5 and wnue1-wnue5</td>
</tr>
<tr>
<td>2nd comm. phase</td>
<td>walfa1-walfa5 (shift right)</td>
</tr>
<tr>
<td>3rd loop nest</td>
<td>compute wg1-wg5 and wtmpl1-wtmp3</td>
</tr>
<tr>
<td>3rd comm. phase</td>
<td>wg1-wg5 and wtmpl1-wtmp3 (shift left)</td>
</tr>
<tr>
<td>4th loop nest</td>
<td>compute wff1-wff5</td>
</tr>
<tr>
<td>4th comm. phase</td>
<td>wff1-wff5 (shift right)</td>
</tr>
<tr>
<td>5th loop nest</td>
<td>produce new results on sr, sm, sp, se, sn &amp; sl</td>
</tr>
</tbody>
</table>

adv3dy with LoopTool [16] for the 1D partitioning. Figure 1 shows a portion of the first and third loop nests in routine adv3dx before and after applying outer loop fusion and array contraction [9]. In adv3dx, variables walfa[12345], wnue[12345], wg[12345], wtmp[123], wff[12345] are defined before being used. Although all these variables are passed as parameters and used in other two routines adv3dy and adv3dz, they are defined locally before their uses in each of the routines. After LoopTool fuses the two outer loop levels, it contracts the original three-dimensional temporary arrays into one-dimensional arrays. Figure 1(b) shows how LoopTool replaces full-size temporary arrays walfa1, wg1, and wtmpl1 by 1D arrays walfa1$, wg1$, and wtmpl1$. LoopTool computes the minimum extent of a contractible array along each dimension based on the distance of associated data dependences and the nesting level of loops carrying these dependences. LoopTool reduces the extent of a dimension whenever possible and eliminates a dimension whenever its extent can be reduced to one.

To create more opportunities for loop fusion and array contraction, LoopTool also reorders statements to bring fusible loops together. To do this safely, it respects control and data dependences as it reorders statements. Figure 1 shows how LoopTool moves the statement defining wra2 from between the two loop nests to before them, which enables the loops to be fused and arrays contracted.

3.3. Exploring Opportunities for Manual Optimization

To investigate opportunities for enhancing the effectiveness of our program optimization tools, we explored manual application of two optimization to improve the performance of IMPACT-3D on microprocessor-based systems. In particular, we investigated hiding the latency of loosely-synchronous communication by overlapping it with computation and eliminating redundant computation to improve node performance.

3.3.1. Efficient Loosely-Synchronous Communication

When compiling IMPACT-3D, dHPF generates code that uses MPI’s asynchronous primitives to implement the loosely-synchronous communication phases needed by the application. adv3d[xyz] execute in several distinct phases: three-dimensional fully parallel loop nests, followed by rounds of near-neighbor communication, in which all communication events complete before starting the next parallel loop. Neither dHPF, nor any other HPF compiler to our knowledge, automatically overlaps loosely-synchronous communication with computation. Therefore, the latency and data transfer time of loosely-synchronous communication are fully exposed to the application. This overhead can be a significant problem when communicating large messages, as for IMPACT-3D with a 1D partitioning.

Figure 2 shows pseudocode for a pair of loop nests separated by loosely-synchronous communication. The first loop nest produces values that are communicated between processors and then consumed by the second loop nest. dHPF generates asynchronous communication operations

```java
do k = 1, l
   do j = 1, m
      do i = 1, n
         produce data that will be sent
         begin receive
         pack data for send
         begin send
         wait for receive & unpack data into shadow regions
         wait for send
         do k = 1, l
            do j = 1, m
               do i = 1, n
                  compute using the data just received

Figure 2. Pseudocode for two loop nests and loosely-synchronous communication in adv3d[xyz].
```
do 10 iz = 1, lz
do 10 iy = 1, ly
do 10 ix = 1, lx-1
  ... walfal(ix,iy,iz) = wdr - wc1
10 continue
wra2 = sram * fstep continue
...

(a) original code

(b) after array contraction

Figure 1. Array contraction in routine adv3dx.

using MPI primitives, but does not currently generate code for loosely-synchronous communication in a way that enables the communication to overlap with the computation of either of the adjacent loop nests.

To evaluate the potential for overlapping loosely-synchronous communication with computation we modified the code generated by dHPF for each of the routines adv3d[xyz], we tiled the loops producing and consuming the communicated data and then adjusted the communication code so that it is performed separately for each tile. After tiling both the computation and the communication, sending data produced by one tile can be overlapped with computation for succeeding tiles. Figure 3 shows pseudocode for two loop nests in adv3d[xyz] that have been tiled. The loosely-synchronous communication phase between them has been tiled as well and scheduled within the tiling loops. In the figure, the operations named begin receive and begin send are asynchronous (the receiving/sending processor does not block until the operation is complete), while the operations named wait for receive and wait for send will block if the data has not been received/sent by the time they are called.

Figure 4 illustrates the the overlapping of the computation and communication for multiple tiles on a 1D partitioned block of data owned by one processor. Each of the four vertical tiles is computed in sequence. When the owning processor completes computation of a data tile, it then sends the portion of the tile needed by a neighbor using an asynchronous MPI operation.

3.3.2. Eliminating redundant computation

In scientific applications, variables may be computed in one phase and recomputed in subsequent phases when the increase in memory footprint for saving the values to be later reused appears more problematic than recomputing them. In IMPACT-3D, redundant computation appears in all three main routines adv3d[xyz]. Figure 5(a) shows a sketch of the first and fourth loop nests in routine adv3dx for each iteration. Thirteen scalar variables \( wu0, wu1, \ldots, whh, wcc \) are computed in the first loop nest and recomputed in the fourth loop nest. The arrays used by loop nests to compute these scalars for each iteration are not modified between the loop nests. Recom-
puting these values might be faster than storing and reloading the values in thirteen additional large 3D arrays. However, if we fuse the two outer loops, 1D arrays suffice to save the values until they are reused. The optimized code after applying fusion and redundant computation elimination is shown in Figure 5(b). In the optimized code, the redundant computation in the fourth loop nest is eliminated. Scalars $w\text{u}0$, $w\text{u}1$, $w\text{uu}$, and $w\text{cc}$ have been expanded to one-dimensional arrays. Nine other scalars involved in this computation (but not shown) are also expanded. Here we applied redundant computation elimination manually. We plan to extend LoopTool to automate this optimization in the future.

(a) original code
(b) after redundant computation elimination

Figure 5. Redundant computation elimination in adv3dx.

Figure 4. Overlapping communication with computation via tiling.

Table 4. 1D BLOCK version with all scalar optimizations.

<table>
<thead>
<tr>
<th># of procs</th>
<th>Rel. Speedup</th>
<th>GFLOPS</th>
<th>% peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.00</td>
<td>35.17</td>
<td>13.74%</td>
</tr>
<tr>
<td>256</td>
<td>1.87</td>
<td>65.68</td>
<td>12.83%</td>
</tr>
<tr>
<td>512</td>
<td>3.26</td>
<td>114.64</td>
<td>11.20%</td>
</tr>
<tr>
<td>1024</td>
<td>5.13</td>
<td>180.39</td>
<td>8.81%</td>
</tr>
</tbody>
</table>

3.4. Effect of Scalar Optimizations

Table 4 shows the effect of applying scalar optimizations (as described in Section 3.2 and 3.3.2) to the adv3d[xy] routines, which operate along non-distributed dimensions ($x$ and $y$) of the 1D partitioned code. The benefits of applying scalar optimizations to reduce the memory footprint increase as we scale up the number of processors. This is because most of the memory footprint in the routines to which we applied array contraction is from the partitioned global arrays, which become smaller on each processor as we scale up the number of processors. Overall, the scalar optimizations improve performance by 21% on 1024 processors of the Alpha server-based Lemieux cluster. Experiments we performed with IMPACT-3D on other microprocessor-based systems, including an Intel Itanium cluster and SGI Origin 2000, have shown as much as a factor of four reduction in cache misses using the scalar optimizations that we describe.
### 3.5. Effect of Overlapping Communication

Table 5 shows the combined effects of both overlapping communication with computation, as described in Section 3.3.1, and applying the scalar optimizations we previously described. As described in Section 3.3.1, the generated code for adv3dz has been hand-modified to better overlap computation with communication. We used a tiling factor of 8 (in the second or y dimension) for the experiments in which communication has been overlapped with computation. Overlapping the communication with computation to hide the data transfer time improves performance by 8% on 1024 processors.

Table 6 presents the results for the experiments conducted using a 3D BLOCK partitioning. We manually eliminated redundant computation and hand-modified the dHPF-generated code for adv3dx, adv3dy and adv3dz to overlap computation with communication, as described in Section 3.3.1. We used a tiling factor of 2 (in the third or z dimension for adv3dx and adv3dy and the second or y dimension for adv3dz). Overlapping communication with computation to hide latency has little impact on the 3D partitioned version. The 3D partitioning uses much smaller messages than the 1D partitioning for which overlapping communication with computation was effective.

### 4. Discussion

We learned two lessons from our experience compiling and tuning IMPACT-3D for clusters of microprocessors. First, traditional loosely-synchronous communication is less efficient as the number of processors increases. We discussed how to improve communication performance in detail in Section 3.3.1. Although we do not have a fully automatic implementation of communication overlap for loosely-synchronous data transfers, we believe our hand optimization can be suitably automated in dHPF. Second, compilers for high-level data parallel languages need a better integration of scalar optimization and data parallelization to achieve high performance on microprocessor-based clusters.

Our current tools perform data parallelization and scalar optimizations separately in an uncoordinated manner. Applying both of them to the same routine is problematic because of the interaction between data and program transformations for scalar optimization and those for parallel code generation.

To understand the interaction we explain how we can currently apply LoopTool transformations to HPF codes:

- **On the HPF source code input.** This is the strategy we used to optimize the implementation of IMPACT-3D with a 1D partitioning. We applied loop and data layout restructuring transformations to the computational routines (adv3dx and adv3dy) operating along the non-distributed dimensions. Since these routines require no communication, the effect of scalar transformations on them does not conflict with dHPF’s parallel code generation. Restructuring the HPF routines that communicate along distributed dimensions is problematic for data parallel compilers: loop fusion and array contraction will alter the possible dependence-based placements for communication events. For example, reducing the number of dimensions of an array can limit the number of loop levels that communication for that array can be vectorized. The resulting generated SPMD code can require element-by-element data transfers inside loops instead of vectorized bulk transfers, which are vastly more efficient.

---

3 We compiled the optimized version of adv3dx (distributing the x dimension) and observed this problem.
• On the resulting SPMD parallelized dHPF output.

The other option for applying scalar and parallelization optimizations in concert is to apply them to dHPF’s generated SPMD code. This strategy has several drawbacks: the parallelizing transformations and communication generation produce new code fragments that were not present in the original global HPF program, including runtime library calls and loops to pack or unpack communicated data. These new code fragments can interfere with loop fusion and array contraction analyses; tracking the use of arrays is much more complex in the SPMD code due to the use of overlap regions for off-processor data, reduced array bounds and more complex loop bounds.

Both approaches are unsatisfactory because neither provides an integrated solution that enables a multidimensional data partitioning to be used effectively in conjunction with scalar optimizations that increase microprocessor efficiency. Using both techniques together requires that they interact and cooperatively choose the shape of the data and loops in the transformed code.

To investigate the potential benefits of scalar transformations when all three dimensions are distributed, we compared the scalar performance of the original version of IMPACT-3D and an optimized version using the problem size 128^3 on a single processor of a local Alpha cluster with the same memory hierarchy as PSC’s Lemieux. Table 7 shows the number of board cache misses, retire delay, and execution cycles of the two versions of codes. The results show that the original version causes about 4 times more cache misses and runs over 20% slower than the optimized version on each processor. Although it is not clear how much overall performance improvement of the parallel code is achievable through scalar optimizations on each processor, these results suggest that a better integration of parallelization and scalar optimization would enable us to improve scalar performance without sacrificing parallel efficiency.

5. Conclusions

We used an experimental HPF compiler in conjunction with a code restructuring tool to explore whether high-level data-parallel programs could be translated into efficient code for microprocessor-based clusters. Our study of the HPF code IMPACT-3D on 128–1024 processors of PSC’s AlphaServer/SC cluster showed that it is possible to achieve good scalability and performance with a high-level data-parallel code on a microprocessor-based cluster.

We achieved an overall performance of 354.17 GFLOPS on 1024 processors of the Lemieux cluster at PSC; this represents 17.29% of the peak performance of those processors. In comparison to the results obtained on the Earth Simulator in 2002, the absolute performance of the AlphaServer cluster is lower due to the smaller number of processors used, as well as the lower performance of each processor (2 GFLOPS vs. 8 GFLOPS). The fraction of peak performance we achieved with IMPACT-3D is considered good for microprocessor-based systems running scientific applications. It is worth noting that, the performance we achieved on a microprocessor-based system with the HPF IMPACT-3D code was higher than the performance achieved by Oliker et al. on three of the four applications they studied at the same scale. (No stronger conclusions can be drawn from this comparison because the results are not directly comparable due to code and platform differences.)

Microprocessor-based clusters lack the prodigious communication bandwidth of the Earth Simulator. For this reason, a one-dimensional parallelization of IMPACT-3D achieved excellent performance and scalability on the Earth Simulator, obtaining good performance and scalability on a microprocessor-based cluster required using a 3D block decomposition rather than a one-dimensional slab decomposition. Changing the data decomposition from 1D to 3D dramatically reduces the surface to volume ratio of the data managed by each processor and significantly reduces the communication volume required in each iteration of the computation.

Our experiments also showed that applying loop and data transformations to reduce the data volume needed by each processor for the computation and to improve memory hierarchy reuse was able to boost performance of the program by 21% on the Lemieux cluster when using a 1D slab decomposition of data. This performance boost was achieved by a combination of loop fusion, eliminating redundant computation, and reducing the size of array temporaries. For our 1D parallelization on Lemieux, these data and loop transformations were applied to the unpartitioned dimensions. In our 3D parallelization, we did not apply these transformations since all data dimensions were partitioned. With careful integration of data and loop transformations into our HPF compiler, it should be possible to apply these transformations in concert with the HPF parallelization to achieve higher performance.

Achieving the best performance on Lemieux using a 1D slab decomposition required overlapping communication and computation. Doing so for a loosely-synchronous application such as IMPACT-3D requires tiling the computational loop producing values being communicated, the communication itself, and the computational loop consuming the communicated values. To our knowledge, transformations of this form have not previously been considered by data-parallel compilers. In this study, we manually edited the code generated by the dHPF compiler to perform this transformation. We are confident that this style of transformation could be automated effectively in data-parallel com-
pilers. At the scale we measured (128-1024 processors), the 3D decomposition does not significantly benefit from overlapping communication and computation in this fashion. However, since partitioning two or more dimensions might not be pragmatic for any given code, this transformation for tolerating communication latency should be part of the repertoire of data-parallel compilers so they can apply it when it will be effective.

Acknowledgements

We thank Y. Seo and H. Sakagami for making the IMPACT-3D source code available to us.

References


