Automatic Cluster Parallelization and Minimizing Communication via Selective Data Replication

Sanket Tavarageri, Benoît Meister, Muthu Baskaran, Benoît Pradelle, Tom Henretty, Athanasios Konstantinidis, Ann Johnson, Richard Lethin

Reservoir Labs, Inc.

HPEC 2015
Outline

1. Overview
2. Automatic Cluster Parallelization
3. Communication Minimization
4. Experimental Evaluation
Overview

- We have designed an integrated compiler and runtime system to map sequential loop nests to clusters.

- **Communication is expensive**: the performance and energy cost of communication is far outpacing the cost of computation.

- We have developed a novel compiler technique to minimize communication between processors.

- The key idea is to **trade memory for communication**.

- Communication energy is reduced by up to 44%.
Communication Performance

Flops are “free”, data motion is expensive

Source: David Scott, Intel at Coalition for Academic Scientific Computation meeting, 2013
Communication Energy

Movement of a 64 bit operand on network costs $\sim$40 times energy of a FLOP!

**Figure**: Source: Shekhar Borkar, Intel, IPDPS 2013
Automatic Cluster Parallelization
R-Stream Parallelization

Input loop

```c
int i;
for (i=0; i<N; i++) {
    A[i] = B[i] + 1;
}
```

Parallelized output for 4 procs

```c
int PROC = r procid();
float A_l[N/4];
float B_l[N/4];
r_dma_get(B_id,(N/4)*PROC, B_l,1,1,N/4);
int i;
for (i = 0; i < N/4; i++) {
    A_l[i] = B_l[i] + 1;
}

r_dma_put(A_l, A_id, (N/4)*PROC, 1, 1, N/4);
```

- `r_dma_get` fetches data
- `r_dma_put` stores data
- `get, put` are implemented using Global Arrays (GAs)
- The **R-Stream DMA engine** emits efficient `dma` instructions: at the task granularity and to contiguous segments of data
A quarter of the array is resident on the local memory of each processor.

- get, puts to a **quarter of the array** will be serviced from local memory, the rest from remote memory.
Communication Minimization
Let’s replicate the array

- A half of the array is resident on the local memory of each processor
- *gets* to a **half of the array** will be serviced from **local memory** as opposed to a quarter before
Automatic data consistency

- A write is propagated to all replicas
Data Replication Considerations

- Data replication reduces communication for gets and increases communication for puts.
- Therefore, it is beneficial when gets (and therefore, reads) in a program are “sufficiently” higher than puts (that is, writes).

**Sufficient Condition for Data Replication**

\[
\frac{(R + W)(N - \beta)}{N} > \frac{R(N - \alpha \beta)}{N} + \frac{\alpha W(N - \beta)}{N}
\]

\[\implies \quad \frac{R}{W} > \frac{N}{\beta} - 1\]

- \(R\): number of reads, \(W\): number of writes, \(N\): number of procs, \(\alpha\): replication factor, \(\beta\): array access pattern correction factor.
- \(R > W\) is the common-case: so, it’s a win!
Inspectors to guide replication

- The number of reads $\mathcal{R}$ and the number of writes $\mathcal{W}$ are found via inspectors

Parallelized output with inspectors

```c
int PROC = r_procid();
// Inspection phase
r_inspector_dma_get(B_id, N/4);
r_inspector_dma_put(A_id, N/4);

r_create_array(FLOATTYPE, A, N, A_id); // replications are made at array
r_create_array(FLOATTYPE, B, N, A_id); // creation time

float A_l[N/4];
float B_l[N/4];
r_dma_get(B_id, (N/4)*PROC, B_l, 1, 1, N/4);
int i;
for (i = 0; i < N/4; i++) {
    A_l[i] = B_l[i] + 1;
}
r_dma_put(A_l, A_id, (N/4)*PROC, 1, 1, N/4);
```
Data replication algorithm

- Arrays are ordered based on their read-to-write ratios

- **Array with the highest** $\frac{R}{W}$ **is replicated first**, then the array with the next highest $\frac{R}{W}$ till memory is available

**Reduction in Remote Memory Accesses**

$$(\alpha - 1)\left(\frac{\beta (R + W)}{N} - W\right)$$

- Higher the data replication factor – $\alpha$, higher will be the reduction in remote memory accesses
Experimental Evaluation
The communication minimization techniques are evaluated in the context of a cluster (8 nodes)

Benchmark set includes: STAP amf (adaptive matched filtering) and covar (covariance estimation) which are part of PERFECT benchmark suite

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Problem size</th>
<th>min $\frac{R}{W}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>corcol (c)</td>
<td>3000 X 3000</td>
<td>2</td>
</tr>
<tr>
<td>gemver (g)</td>
<td>3000 X 3000</td>
<td>3</td>
</tr>
<tr>
<td>doitgen (d)</td>
<td>50 X 50 X 50</td>
<td>2</td>
</tr>
<tr>
<td>planck (p)</td>
<td>5000</td>
<td>2</td>
</tr>
<tr>
<td>hydro (h)</td>
<td>2000 X 2000</td>
<td>3</td>
</tr>
<tr>
<td>RTM (r)</td>
<td>264 X 264 X 264</td>
<td>2</td>
</tr>
<tr>
<td>amf (a)</td>
<td>4 X 512 X 32</td>
<td>2</td>
</tr>
<tr>
<td>covar (v)</td>
<td>4 X 512 X 32</td>
<td>4</td>
</tr>
</tbody>
</table>
Communication minimization

**Figure: Local accesses**
- Local memory accesses are up 1.93X (geometric mean)
- Remote memory accesses are reduced by up to 47% and by 15.5% on average

**Figure: Data movement between nodes**
Performance and energy improvements

Figure: Performance of communication minimizing codes

- Performance increases 1.6% on average (geometric mean)
- Communication energy is reduced by up to 44% and 14% on average

Figure: Estimated communication energy
Related Work

Compared to Demmel’s work [Solomonik, 2011], it is more general:

1. It is not algorithm-specific
2. Even write data may be replicated and data consistency is automatically maintained
3. It is applicable to all processor grid configurations and not just 2.5D

Unlike the related polyhedral communication code generation schemes [Dathathri, 2013], our method:

1. Is more flexible. For example, the data may be replicated to different degrees while using the same generated code
2. Is more scalable – if the communication code were to be baked into the compiled code, the code becomes bulky affecting performance.
References

E. Solomonik and J. Demmel
Communication-optimal parallel 2.5 D matrix multiplication and LU factorization algorithms

Euro-Par Parallel Processing 2011.

Generating efficient data movement code for heterogeneous architectures with distributed-memory

22nd International Conference on Parallel Architectures and Compilation Techniques (PACT) 2013.