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

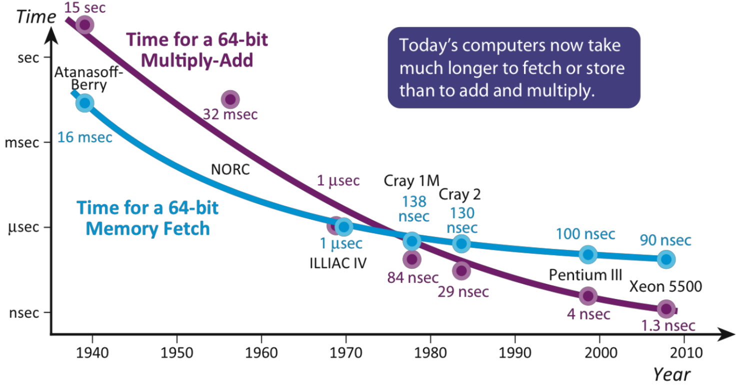
1. Overview
2. Automatic Cluster Parallelization
3. Communication Minimization
4. Experimental Evaluation

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

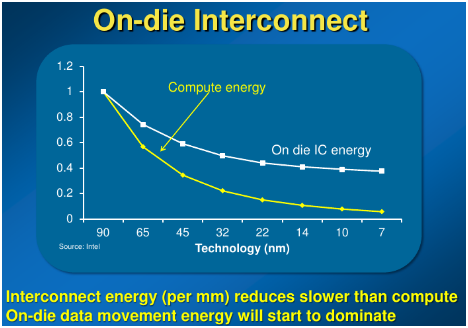
Flops are “free”, data motion is expensive



Today’s computers now take much longer to fetch or store than to add and multiply.

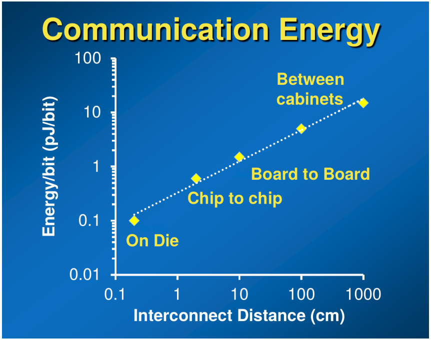
Source: David Scott, Intel at Coalition for Academic Scientific Computation meeting, 2013

Communication Energy



Interconnect energy (per mm) reduces slower than compute
On-die data movement energy will start to dominate

Figure : Source: Shekhar Borkar, Intel, IPDPS 2013



- Movement of a 64 bit operand on network costs ∼40 times energy of a FLOP!

Automatic Cluster Parallelization

R-Stream Parallelization

Input loop

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

Parallelized output for 4 procs

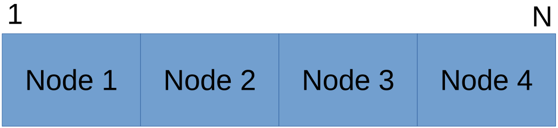
```
int  PROC  =  r_procid();
float  A_l[N/4];
float  B_l[N/4];
r_dma_get(B_id,(N/4)*PROC,B_l,1,1,N/4);
int  i;
for  (i  =  0;  i  <  N/4;  i++) {
     A_l[i]  =  B_l[i]  +  1;
}

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

Reservoir Labs

Distribution of an array in global address space

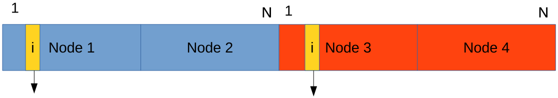


- A quarter of the array is resident on the local memory of each processor
- get, puts to a **quarter of the array** will be serviced from local memory, the rest from remote memory

Reservoir Labs

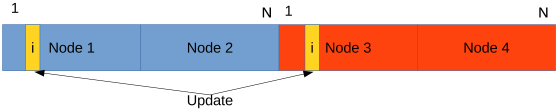
Communication Minimization

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

$$\frac{(\mathcal{R}+\mathcal{W})(\mathcal{N}-\beta)}{\mathcal{N}} > \frac{\mathcal{R}(\mathcal{N}-\alpha\beta)}{\mathcal{N}} + \frac{\alpha\mathcal{W}(\mathcal{N}-\beta)}{\mathcal{N}}$$

$$\implies \qquad \frac{\mathcal{R}}{\mathcal{W}} > \frac{\mathcal{N}}{\beta} - 1$$

- $\mathcal{R}$: number of reads, $\mathcal{W}$: number of writes, $\mathcal{N}$: number of procs, α : replication factor, β: array access pattern correction factor
- $\mathcal{R} > \mathcal{W}$ 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

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

Reduction in Remote Memory Accesses

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

- Higher the data replication factor – α, higher will be the reduction in remote memory accesses

Experimental Evaluation

Set up

- 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 : Benchmarks

Benchmark	Problem size	min $\frac{\mathcal{R}}{\mathcal{W}}$
corcol (c)	3000 X 3000	2
gemver (g)	3000 X 3000	3
doitgen (d)	50 X 50 X 50	2
planck (p)	5000	2
hydro (h)	2000 X 2000	3
RTM (r)	264 X 264 X 264	2
amf (a)	4 X 512 X 32	2
covar (v)	4 X 512 X 32	4

Communication minimization

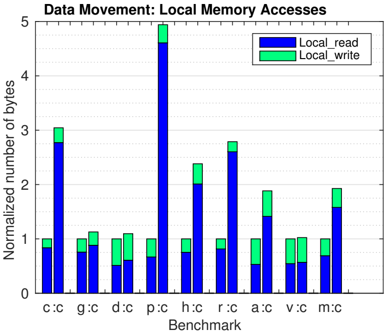


Figure : Local accesses

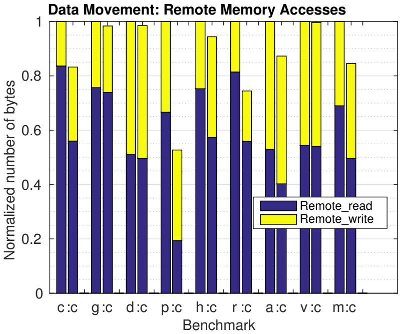


Figure : Data movement between nodes

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

Performance and energy improvements

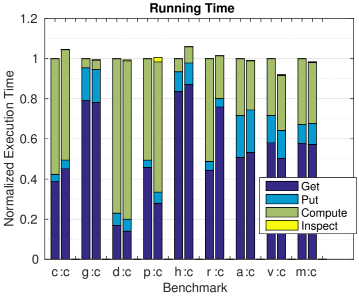


Figure : Performance of communication minimizing codes

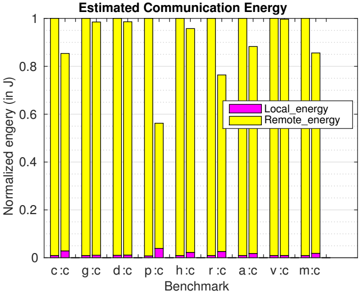


Figure : Estimated communication energy

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

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.

R. Dathathri, C. Reddy, T. Ramashekar, and U. Bondhugula
Generating efficient data movement code for heterogeneous architectures with distributed-memory
22nd International Conference on Parallel Architectures and Compilation Techniques (PACT) 2013.