Lecture 11

Memory hierarchy performance
Advanced Collective Communication
Announcements

- No class on 11/17 and 11/19
- Makeup lectures
  - Friday 11/20. 3:00 to 4:20
  - Weds 12/2  5-7PM
- CSE 260 Symposium
  - Week 10: Tues, Weds, Thurs
- Assignment return and feedback
• Use single precision
• Baseline can be a single CPU core, but in that case I have raised the bar
  – \times 10 performance improvement for groups of 2
  – \times 20 for groups of 3
  – Use a problem of size 512^3
• Report timing both with and without transfer between device and host
Measuring performance

• Two ways
• Use Cuda events/elapsed time
• Use an ordinary timer, e.g. gettimeofday()
• See $CUDA_Examples/withTimer
• Note that kernel invocation is asynchronous

```c
cudaThreadSynchronize();
double t_device_compute = -getTime();

// COMPUTE
cudaThreadSynchronize();
t_device_compute += getTime();
```
CUDA Error Handling

• Cuda can silently fail, you can observe misleading performance
• E.g. if you specify an invalid grid / thread block dimensions
• Beware that the last error can be cleared by successive kernel calls, so check frequently

```c
assert(cudaSuccess == cudaMalloc((void **) &a_d, size));
printf("Cuda error: %s\n", cudaGetErrorString(cudaGetLastError()));
```

• What about asynchronous calls?
• cf CUDA Programming Guide, “Error Handling”
Memory Performance Issues in Stencil Methods

- Processor Geometry
- Cache misses – conflicts
- Sharing among threads
- False sharing
Stencil Memory access patterns

Rivera and Tseng
Memory Strides

H. Das, S. Pan, L. Chen

Sam Williams et al.
Stencil Patterns in memory
Processor Geometry

![Graph showing the relationship between processor geometry and speedup across different configurations. The x-axis represents processor geometry, and the y-axis represents speedup. Five different line styles are used to represent various values of N, ranging from N=256 to N=808.]
Interactions between threads
Interactions between threads

- Thread sharing
- False sharing
Cache Conflicts

Rivera and Tseng
Acceleration
Performance

• For a problem that fits in L3, but not in L2 (or L1)
  \[56^3 = 56 \times 56 \times 56 \times 8 \times 3 = 4,214,784\]

• 
  ```
  ./jac3d -N 56 -i 1000 -px X -py Y -pz Z
  ```
  – \{1,1,1\}: 0.813 Gflops
  – \{1,2,1\}: 1.581 Gflops
  – \{1,4,1\}: 3.084 Gflops
  – \{1,4,2\}: 4.711 Gflops
Collective Communication
Collective communication

- Arises in many applications
  - Fast Fourier Transform
  - Sorting
- Collective operations are called by all processes within a communicator
- Simplest collectives
  - Broadcast: distribute data from a designated root process to all the others
  - Reduce: combine data from all processes returning the result to the root process
- Other Useful collectives
  - Scatter/gather
  - All to all
  - Allgather
Underlying assumptions

- Fast interconnect structure
  - All nodes are equidistant
  - Single-ported, bidirectional links
- Communication time is $\alpha + \beta n$ in the absence of contention
  - Determined by bandwidth $\beta^{-1}$ for long messages
  - Dominated by latency $\alpha$ for short messages
Inside MPI-CH

- Tree like algorithm to broadcast the message to blocks of processes, and a linear algorithm to broadcast the message within each block
- Block size may be configured at installation time
- If there is hardware support, then it is given responsibility to carry out the broadcast
- Polyalgorithms apply different algorithms to different cases, i.e. long vs. short messages, different machine configurations
- We’ll use hypercube algorithms to simplify the special cases when $P=2^k$, $k$ an integer
Details of the algorithms

• Scatter/gather
• All to all
• Allgather
• Revisiting broadcast
Scatter/Gather family

$P_0$  $P_1$  $P_{p-1}$

Gather

Scatter

Root
Scatter

• Simple linear algorithm
  – Each processor sends a chunk of data to all others
  – Reasonable for long messages

\[(p - 1)\alpha + \frac{p - 1}{p} n\beta\]

• Similar approach taken for Reduce and Gather
• For short messages, we need to reduce the complexity of the latency (\(\alpha\)) term
Minimal spanning tree algorithm

• Recursive hypercube-like algorithm with \([ \log P \) steps
  ➢ Root sends half its data to process \((\text{root} + \frac{p}{2}) \mod p\)
  ➢ Each receiver acts as a root for corresponding half of the processes

• Running time: \(\lceil \log P \rceil \alpha + \frac{p - 1}{p} n \beta\)

• Requires \(O(n/2)\) buffer space
Details of the algorithms

• Scatter/gather
• All to all
• Allgather
• Revisiting broadcast
All to all

- Also called total exchange or personalized communication: a transpose
- Each process sends a different chunk of data to each of the other processes
- Used in sorting and the Fast Fourier Transform
Exchange algorithm

- $n$ elements / processor ($n$ total elements)
- $p - 1$ step algorithm
  - Each processor exchanges $n/p$ elements with each of the others
  - In step $i$, process $k$ exchanges with processes $k \pm i$

$$\text{for } i = 1 \text{ to } p-1$$
$$\quad \text{src} = (\text{rank} - i + p) \mod p$$
$$\quad \text{dest} = (\text{rank} + i ) \mod p$$
$$\quad \text{sendrecv( from src to dest )}$$
$$\text{end for}$$

- Good algorithm for long messages
- Running time:
  $$\left( p - 1 \right) \alpha + \left( p - 1 \right) \frac{n}{p} \beta \approx n \beta$$
Recursive doubling for short messages

• In each of \([\log p]\) phases all nodes exchange \(\frac{1}{2}\) their accumulated data with the others

• Only \(P/2\) messages are sent at any one time

\[
D = 1 \\
\text{while } (D < p) \\
\quad \text{Exchange & accumulate data with rank } \otimes D \\
\quad \text{Left shift } D \text{ by 1}
\]

end while

• Optimal running time for short messages

\[
\lfloor \lg P \rfloor \alpha + nP \beta \approx \lfloor \lg P \rfloor \alpha
\]
Flow of information
Flow of information

A B C D ←→ A B C D

10

00

11

01

P0  P1  P2  P3

Scott B. Baden / CSE 260 / Fall 2009
Flow of information

A B C D

10 11

00 01

A B C D

P0 P1 P2 P3

A B C D
Summarizing all to all

- Short messages $\lceil \log P \rceil \alpha$
- Long messages $\frac{P-1}{P} n\beta$
Details of the algorithms

• Scatter/gather
• All to all
• **Allgather**
• Revisiting broadcast
AllGather

- Equivalent to a gather followed by a broadcast
- All processors accumulate a chunk of data from all the others
AllGather

P_0  P_1  P_{p-1}
Allgather

- Use the all to all recursive doubling algorithm
- For $P$ a power of two, running time is
  \[
  [\lg P]\alpha + \frac{p - 1}{p} n\beta
  \]

- Also used in Allgatherv
“Vector” variants

• Generalize all-to-all, gather, etc.
• Processes supply varying length datum
• Vector all-to-all

\[ \texttt{MPI\_Alltoallv} ( \]
\[ \text{void \}*sendbuf, \text{int \}sendcounts[], \text{int \}sDispl []}, \]
\[ \text{MPI\_Datatype \}sendtype,} \]
\[ \text{void\}* \)recvbuf, \text{int \}recvcounts[], \text{int \}rDispl[]}, \]
\[ \text{MPI\_Datatype \}recvtype, MPI\_Comm \)comm ) \]
Alltoallv used in sample sort

Details of the algorithms

• Scatter/gather
• All to all
• Allgather
• Revisiting broadcast
Revisiting Broadcast

• P may not be a power of 2
• We use a binomial tree algorithm
• We’ll use the hypercube algorithm to illustrate the special case of $P=2^k$
• Hypercube algorithm is efficient for short messages
• We use a different algorithm for long messages
Strategy for long messages

• Based van de Geijn’s strategy
• Scatter the data
  – Divide the data to be broadcast into pieces, and fill the machine with the pieces
• Do an Allgather
  – Now that everyone has a part of the entire result, collect on all processors
• Faster than MST algorithm for long messages

\[ 2 \frac{p - 1}{p} n \beta \ll \lfloor \lg p \rfloor n \beta \]
Algorithm for long messages

The scatter step

$P_0$  $P_1$  $P_{p-1}$  Root

Scatter
Algorithm for long messages

AllGather step
References

