Matrix-Matrix Multiplication

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Due: 05/11/16
Posted: 05/02/16
Updated: 05/01/16
Table of Contents

1 Quiz #4, Comparing MPI and CUDA Matrix-Matrix Multiplication
2 General Instructions
3 General: Comparison of different parallel models for estimating PI
4 Examples for comparing MPI and CUDA
Comparing an MPI and CUDA Matrix-Matrix Multiplication

Objective:
- Develop and test a CUDA Matrix-Matrix Multiplication code
- Compare your CUDA results to MPI reference data
- All input and source code can be found in /COMP605/quiz4/

Serial mat-mat-mult: a copy has been place in quiz4 directory, and you can see Pacheco Parallel Programmins with MPI, Ch7.

MPI mat-mat-mult:
- you do not need to run any MPI code
- you can use the reference data provided in the quiz directory

CUDA mat-mat-mult:
- Your may write your own code, or modify existing code.
- Working CUDA source code has been provided in the quiz directory:
  - CUDA Toolkit:
    - http://docs.nvidia.com/cuda/cuda-samples/index.html#axzz47VXhJJc2
  - Nitin Gupta (Nvidia developer):

Be sure to clearly reference code sources both in the report and in the code.
Programming Instructions

- Two test types:
  - Read A & B from input files (small test cases)
  - Generate input matrices A & B from within code
- All key variables and filenames read from command line
- Matrix size $N$ and allocations should be dynamic
- Vary #threads/block for a given $N$ (see Figure 2 below).
- Use cuda properties to check that your matrix fits on the device and to set the device
- All jobs should be run using batch scripts
- For *small* test cases ($< 128$), save matrices A, B, & C to separate files (not in the queue output file).
Vary the size of the matrices using square \([N_i \times N_j]\) matrices

Vary #CUDA threads: use square grid/block/thread distribution

Recall: GPU hardware limits the number of blocks per grid and the number of threads per block

Larger problems require use of both grid and blocks

Need to control the number of threads, since they are smaller

Fix number of threads and distributed chunks along the blocks:

\[
\begin{align*}
\text{add} & \left\langle\langle 128, 128 \rangle\right\rangle (\text{dev}_a, \text{dev}_b, \text{dev}_c); \\
\text{add} & \left\langle\langle h_N, h_N \rangle\right\rangle (\text{dev}_a, \text{dev}_b, \text{dev}_c); \\
\text{add} & \left\langle\langle \text{ceil}(h_N/128), 128 \rangle\right\rangle (\text{dev}_a, \text{dev}_b, \text{dev}_c); \\
\text{add} & \left\langle\langle (h_N+127)/128, 128 \rangle\right\rangle (\text{dev}_a, \text{dev}_b, \text{dev}_c);
\end{align*}
\]

if \(\text{maxTh} ==\) maximum number of threads per block:

\[
\text{add} \left\langle\langle (h_N+(\text{maxTh})-1)/\text{maxTh}, \text{maxTh} \rangle\right\rangle (\text{dev}_a, \text{dev}_b, \text{dev}_c);
\]

Compute thread index as:

\[
tid = \text{threadIdx.x} + \text{blockIdx.x} \times \text{blockDim.x};
\]

\[
\$tid = \text{threadIdx.x} + \text{blockIdx.x} \times \text{blockDim.x};$
\]
Performance

- keep track of what node/core you used (set the device)
- Timing:
  - Time code runs ($T_{wall}$)
  - Twall and Speedup
  - Compare MPI CPU to GPU versions
- What is the largest #threads you were able to test? What happened, why do you think this happened
Suggestions for comparing MPI and CUDA

- You cannot directly compare the MPI PEs/cores against the OpenMP or CUDA number of threads.
- You can compare common run-time characteristics and variables:
  - All runs can have same (or close) problem sizes
  - All runs can be timed
- You can find $T_{\text{optimal}}$ for each model:
  
  \textit{defined as the point where increasing the number of processors or the number of threads/block no longer significantly reduces the run-time ('turnover' point).}
Suggestions on what to Report/Turn in for both problems:

- Create the homework directory `USER/hw/q4` with correct access permissions.
- Short lab report with comments, figures and table labels.
- Explain your results for Thread and ProbSize scaling.
- Include relevant tables of your test data.
- Evidence you ran your jobs using the batch queue (short/small job); examples of batch scripts.
- Plots of key results.
- A copy of your code (single spaced, two sided, two column format is OK).
- Reference key sources of information *in your report and code* where applicable.
Examples for comparing MPI and CUDA

- The figures below show how to identify \( T_{optimal} \) for MPI and CUDA programming models.
- Figures 1 and 2 show examples of how to determine \( T_{optimal} \) for the MPI and CUDA programming models.
- Figure 3 shows a way to compare the two programming models based on \( T_{optimal} \).
- Figures 4-6 show plots of the MPI reference data provided for the assignment.
- Note: note all data is \textit{not} for timings mat-mat-mul, nor run on the same cores as are currently on tuckoo, so your data values may differ, but the trends should not change.
Figure 1: The figure above shows the run-time as a function of the number of processors, for different problem sizes, using MPI. The run time decreases as the number of cores increases, up to a limit where there is not much improvement. In this case, $T_{optimal}$ 16 cores
Example Comparing MPI and CUDA

Figure 2: $T_{wall}$ for different $N_{threads/block}$ vs $Dim$. The figure above shows the run-time as a function of the number of threads per block, for different problem sizes, using CUDA. The run time decreases as the number of threads per block increases, up to a limit where there is not much improvement. In this case, $T_{optimal} = 64$ threads/block.

Source: Fall 2012: G. Pham
Figure 3: $T_{optimal}$ as a function of matrix size for MPI and CUDA/GPU tests. The figure above shows that for a given problem size, $T_{optimal}$ for the GPU programming model is better than MPI. This problem is for a matrix-matrix multiplication problem.

Source: Fall 2012: G. Pham
### MPI Matrix-Matrix Multiplication ref data

<table>
<thead>
<tr>
<th>Cores</th>
<th>420</th>
<th>1260</th>
<th>1680</th>
<th>2520</th>
<th>3360</th>
<th>4200</th>
<th>5040</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.78E-01</td>
<td>2.56E+01</td>
<td>6.91E+01</td>
<td>2.34E+02</td>
<td>4.98E+02</td>
<td>1.09E+03</td>
<td>1.88E+03</td>
</tr>
<tr>
<td>4</td>
<td>1.96E-01</td>
<td>5.61E+00</td>
<td>1.37E+01</td>
<td>5.85E+01</td>
<td>1.37E+02</td>
<td>2.71E+02</td>
<td>4.66E+02</td>
</tr>
<tr>
<td>9</td>
<td>1.10E-01</td>
<td>2.35E+00</td>
<td>6.84E+00</td>
<td>2.39E+01</td>
<td>5.91E+01</td>
<td>1.25E+02</td>
<td>2.10E+02</td>
</tr>
<tr>
<td>16</td>
<td>6.28E-02</td>
<td>1.64E+00</td>
<td>3.68E+00</td>
<td>1.38E+01</td>
<td>3.30E+01</td>
<td>8.05E+01</td>
<td>1.39E+02</td>
</tr>
<tr>
<td>25</td>
<td>4.88E-02</td>
<td>1.45E+00</td>
<td>3.52E+00</td>
<td>8.23E+00</td>
<td>3.00E+01</td>
<td>6.79E+01</td>
<td>8.67E+01</td>
</tr>
<tr>
<td>36</td>
<td>9.40E-02</td>
<td>9.70E-01</td>
<td>2.43E+00</td>
<td>8.23E+00</td>
<td>2.20E+01</td>
<td>4.08E+01</td>
<td>7.58E+01</td>
</tr>
<tr>
<td>49</td>
<td>4.01E-02</td>
<td>7.08E-01</td>
<td>1.98E+00</td>
<td>4.59E+00</td>
<td>1.25E+01</td>
<td>2.49E+01</td>
<td>3.99E+01</td>
</tr>
</tbody>
</table>

**Figure 4:** MPI Matrix-Matrix Multiplication ref data. The Table shows the runtime (in seconds) as a function of the number of processors for different matrix sizes.
Figure 5: MPI Matrix-Matrix Multiplication ref data: Curves show the runtime (in seconds) as a function of the matrix size for different number processors.
Figure 6: MPI Matrix-Matrix Multiplication ref data: Curves show the runtime (in seconds) as a function of the number of processors for different matrix sizes.