Performance Tool Integration in Programming Environments for GPU Acceleration: Experiences with TAU and HMPP

Allen D. Malony
Shankar Mayanglambam, Sameer Shende, Matt Sottile

Introduction

Parallel programming environments targeting GPU accelerators hide the complexity of working with raw devices by allowing the application developer to work with libraries, special language constructs, or directives to a compiler. The benefit for the programmer is a higher-level abstraction for accelerator programming and protection of their software investment, since the environment takes the responsibility for translating the program to work with different accelerator backends.

The challenge for accelerator programming environments is to provide high-level support and flexibility without sacrificing delivered performance. For optimization of GPU-accelerated applications, tools must 1) be able to measure performance of GPU computations, and 2) be integrated with the high-level programming framework to generate important performance events and meta data for representing performance results to the user.

We have developed an approach (called TAUcuda) to measure the performance of GPU computations programmed using CUDA and integrate this information with application performance data captured with the TAU Performance System. To address the high-level programming aspect, we have integrated TAU/TAUcuda with the HMPP Workbench. The design methodology includes an instrumentation strategy whereby HMPP automatically inserts calls to the TAU/TAUcuda measurement interfaces in its runtime system and HMPP-compiled code to capture a performance picture of the resulting application execution.

TAUcuda Methodology

void tau_cuda_init(int argc, char **argv);
To be called when the application starts
Initializes data structures and checks GPU status
void tau_cuda_exit();
To be called before any thread exits at end of application
All the CUDA profile data output for each thread of execution
void tau_cuda_stream_begin(double *event, cudaStream_t stream);
Called before CUDA statements to be measured
Returns handle which should be used in the end call
void tau_cuda_stream_end(void *handle);
Create new CUDA event profile object is created
void tau_cuda_finalize(cudaStream_t stream);
Called immediately after CUDA statements to be measured
Inserts a CUDA event into the stream identified by the handle

One Stream Tests

<table>
<thead>
<tr>
<th>CPU Load</th>
<th>GPU Load</th>
<th>Event</th>
<th>Initial Exclusive Time</th>
<th>Exclusive Time</th>
<th>Wait Time</th>
<th>Finalize Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Interpolate</td>
<td>7322.4,52,922</td>
<td>7322.4922</td>
<td>7513.4926</td>
<td>87.4906</td>
</tr>
<tr>
<td>3</td>
<td>2X</td>
<td>Interpolate</td>
<td>2580.9,52,922</td>
<td>2580.9922</td>
<td>3217.9926</td>
<td>87.4906</td>
</tr>
<tr>
<td>3</td>
<td>3X</td>
<td>Interpolate</td>
<td>2580.9,52,922</td>
<td>2580.9922</td>
<td>3217.9926</td>
<td>87.4906</td>
</tr>
<tr>
<td>3</td>
<td>3Y</td>
<td>Interpolate</td>
<td>74085.9,52,922</td>
<td>74085.9922</td>
<td>46097.9926</td>
<td>87.4906</td>
</tr>
<tr>
<td>10Y</td>
<td>X</td>
<td>Interpolate</td>
<td>75086,52,922</td>
<td>75086.9922</td>
<td>42563.9926</td>
<td>82498.6289</td>
</tr>
</tbody>
</table>

Game of Life Performance

Matrix-Vector Multiply

Two codelets allow overlap of data transfer and computation
Demonstrates profiling and tracing

This work was supported by a 2009 NVIDIA Professor Award to Prof. Allen D. Malony.