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

Allen D. Malony[1,2], **Shangkar Mayanglambam**[1], **Sameer Shende**[1,2], **Matt Sottile**[1]
[1]Computer and Information Science Department, University of Oregon, [2]ParaTools, Inc.

Laurent Morin, Stephane Bihan, Francois Bodin
CAPS Entreprise

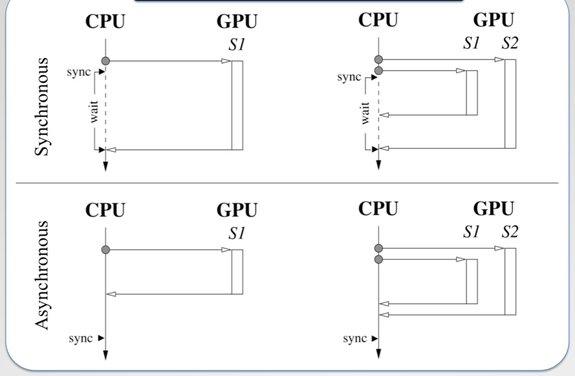
Introduction

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 acceleration backends.

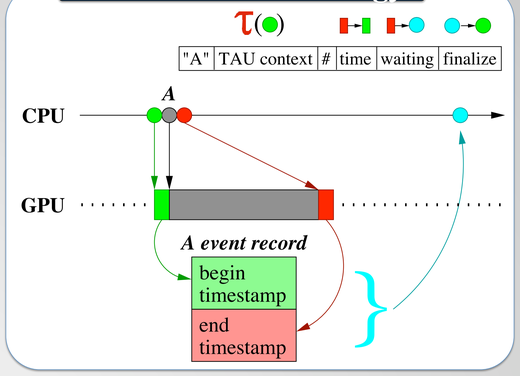
The challenge for accelerator programming environments is to provide high-level support and flexibility without sacrificing delivered performance. For optimization of GPU-accelerated applications, tools must 1) be able to measure performance of GPU computations, and 2) be integrated with the high-level 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-translated code to capture a performance picture of the resulting application execution.

CPU-GPU Scenarios



TAUcuda Methodology



TAUcuda API

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(char *event, cudaStream_t stream);
- Called before CUDA statements to be measured
- Returns handle which should be used in the end call
- Create new CUDA event profile object is created

void tau_cuda_stream_end(void * handle);
- Called immediately after CUDA statements to be measured
- Inserts a CUDA event into the stream identified by the handle

vector<Event> tau_cuda_update();
- Checks for completed CUDA events on all streams
- Non-blocking and returns # completed on each stream

int tau_cuda_update(cudaStream_t stream);
- Same as *tau_cuda_update*() except for a particular stream
- Non-blocking and returns # completed on the stream

vector<Event> tau_cuda_finalize();
- Waits for all CUDA events to complete on all streams
- Blocking and returns # completed on each stream

int tau_cuda_finalize(cudaStream_t stream);
- Same as *tau_cuda_finalize*() except for a particular stream
- Blocking and returns # completed on the stream

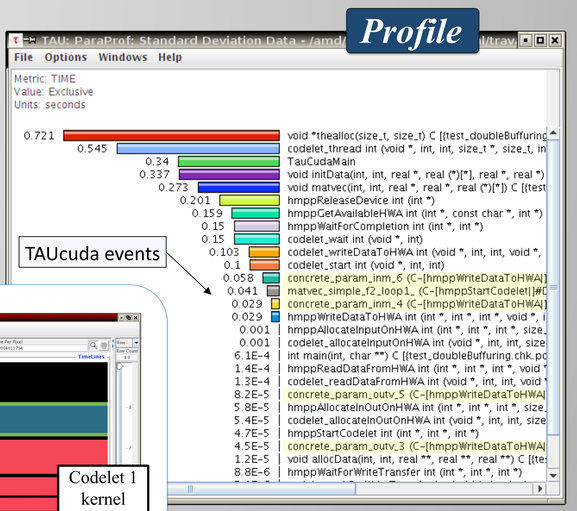
One Stream Tests

CPU Load	GPU Load	Event	Inclusive Time	Exclusive Time	Wait Time	Finalize Time
0	X	Interpolate (C-[main—]#D-0#S-0)	75222.4922	75222.4922	75134.7656	87.8906
0	2X	Interpolate (C-[main—]#D-0#S-0)	150097.7031	150097.7031	149995.6094	102.0508
0	3X	Interpolate (C-[main—]#D-0#S-0)	225034.2031	225034.2031	224915.5312	118.6523
Y	X	Interpolate (C-[main—]#D-0#S-0)	74985.6953	74985.6953	64097.1680	10888.6719
2Y	X	Interpolate (C-[main—]#D-0#S-0)	75058.5234	75058.5234	42563.9648	32494.6289
10Y	X	Interpolate (C-[main—]#D-0#S-0)	75032.9609	75032.9609	0.0000	108114.7500

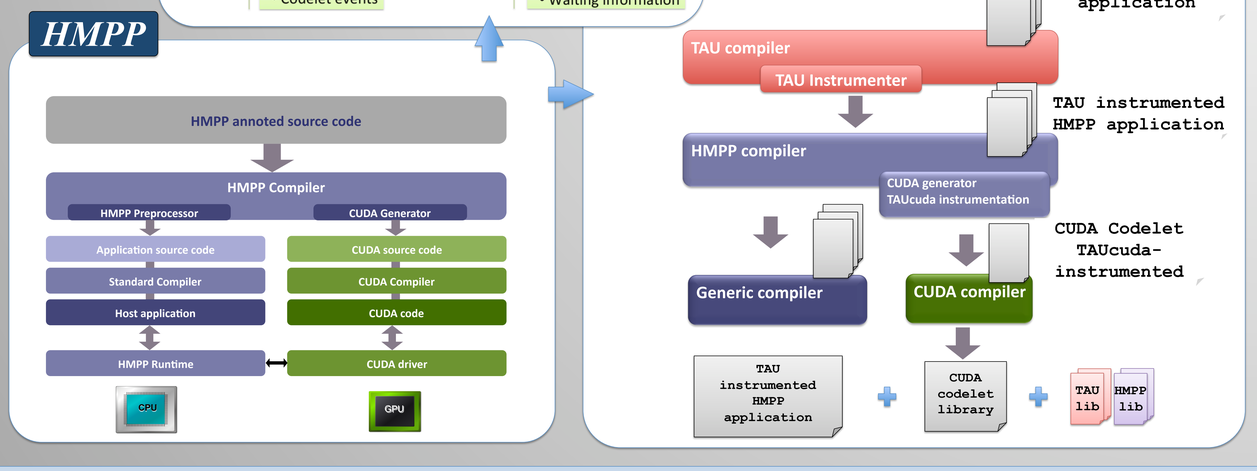
Game of Life Performance

TAU Event	calls	Computation time(ms) for varying input matrix size				
		1000X1000	2000X2000	3000X3000	4000X4000	5000X5000
hmppStartCodelet	1	7931	25506	95711	152348	345741
hmppGetAvailableHWA	1	2765859	2702177	2705051	2769246	2714853
hmppReleaseDevice	1	50235	2732	2831	44236	2923
hmppWriteDataToHWA	10	1318	1596	2064	2694	3488
hmppAllocateInputOnHWA	7	1234	1235	1234	1241	1277
hmppReadDataFromHWA	3	1718	3261	5648	8894	12811
hmppAllocateInOutOnHWA	3	1218	1357	1256	1302	1277
hmppStartHMPP	1	9	11	11	12	11
codelet_wait	1	5566	22925	93184	149688	343263
codelet_readDataFromHWA	3	590	2021	4320	7522	11534
codelet_writeDataToHWA	10	121	371	808	1430	2235
codelet_allocateInOutOnHWA	3	81	88	99	117	138
codelet_start	1	104	127	134	131	131
codelet_allocateInputOnHWA	7	0	0	1	1	1

Profile



HMPP-TAU



HMPP

Trace

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 Partner award to Prof. Allen D. Malony.

ParaTools

http://tau.uoregon.edu
http://caps-entreprise.com