

HPCToolkit Performance Tools

Performance analysis of CPU and GPU-accelerated applications

John Mellor-Crummey Professor, Rice University



HPCToolkit Funding Acknowledgments

- Government
 - -Exascale Computing Project 17-SC-20-SC
 - -Lawrence Livermore National Laboratory Subcontract B645220
 - -Argonne National Laboratory Subcontract 9F-60073
- Corporate
 - -Advanced Micro Devices
 - -Intel Corporation
 - -TotalEnergies EP Research & Technology USA, LLC.





Rice University's HPCToolkit Performance Tools

Measure and analyze performance of CPU and GPU-accelerated applications

- Easy: profile unmodified application binaries
- Fast: low-overhead measurement
- · Informative: understand where an application spends its time and why
 - -call path profiles associate metrics with application source code contexts
 - -optional hierarchical traces to understand execution dynamics
- Broad audience
 - -application developers
 - -framework developers
 - -runtime and tool developers





HPCToolkit's Workflow for CPU Applications

























Measurement of CPU and GPU-accelerated Applications

- Sampling using timers and hardware counter overflow on the CPU
- Callbacks when GPU operations are launched and (sometimes) completed
- GPU event stream for GPU operations; PC Samples (NVIDIA)





Call Stack Unwinding to Attribute Costs in Context

- Unwind when timer or hardware counter overflows
 - -measurement overhead proportional to sampling frequency rather than call frequency
- Unwind to capture context for events such as GPU kernel launches



Calling context tree







hpcrun: Measure CPU and/or GPU activity

- GPU profiling
- GPU instrumentation (Intel GPU only) —hpcrun -e gpu=level0, inst=count, latency <app>
- GPU PC sampling (NVIDIA GPU only) —hpcrun -e gpu=nvidia,pc <app>
- CPU and GPU Tracing (in addition to profiling)
 —hpcrun -e CPUTIME -e gpu=xxx -t <app>
- Use hpcrun with job launchers — jsrun -n 32 -g 1 -a 1 hpcrun -e gpu=xxx <app> — srun -n 1 -G 1 hpcrun -e gpu=xxx <app> — aprun -n 16 -N 8 -d 8 hpcrun -e gpu=xxx <app>

xxx E {nvidia,amd,opencl,level0}

Profiles: aggregated on the fly

- a calling context tree per thread
- a calling context tree per GPU stream
- instruction level measurements

CPU traces

- trace of call stack samples

GPU traces

- trace of call stacks that initiate GPU operations











hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

• Usage

```
hpcstruct [--gpucfg yes] <measurement-directory>
```

- What it does
 - Recover program structure information
 - Files, functions, inlined templates or functions, loops, source lines
 - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
 - -default: use size(CPU set)/2 threads
 - —analyze large application binaries with 16 threads
 - -analyze multiple small application binaries concurrently with 2 threads each
 - Cache binary analysis results for reuse when analyzing other executions

NOTE: --gpucfg yes needed only for analysis of GPU binaries when NVIDIA PC samples were collected





Step 4:

•







hpcprof/hpcprof-mpi: Associate Measurements with Program Structure

- Analyze data from modest executions sequentially hpcprof <measurement-directory>
- Analyze data from large executions in parallel

jsrun -n 32 -a 1 hpcprof-mpi <measurement-directory> srun -n 32 hpcprof-mpi <measurement-directory> aprun -n 128 -N 8 hpcprof-mpi <measurement-directory>





Step 4:







Code-centric Analysis with hpcviewer







Understanding Temporal Behavior

- Profiling compresses out the temporal dimension
 - -Temporal patterns, e.g. serial sections and dynamic load imbalance are invisible in profiles
- What can we do? Trace call path samples
 - -N times per second, take a call path sample of each thread
 - -Organize the samples for each thread along a time line
 - -View how the execution evolves left to right
 - —What do we view? assign each procedure a color; view a depth slice of an execution







Time-centric Analysis with hpcviewer

stream

GPU

ranks, OpenMP Threads,

ЧЫ



zonne 🍊

Time-centric Analysis with hpctraceviewer

Experimental version of QMCPack

- 32 ranks
- 32 threads each

Summary view summarizes activity across threads at each point in time







hpcstruct Example: Analyze 7.7GB TensorFlow library (170MB text) in 77s

hpcviewer	
File View Filter Help	
📴 Profile: hpcstruct-bin 國 Trace: hpcstruct-bin	🙆 💙 😂 🏠 🔶 🏟 🕶 💾 💆 🖻 🗖
Main view	Depth: 496 - +
Time Range: [0s, 77s] Rank Range: [Thread 0, Thread 31] Cross Hair: (53s, Thread	
	Call stack Statistics
	realmain(int, char**)
	doSingleBinary(Args&, stat*)
	BAnal::Struct::makeStructure(std::_cxxll::b.
	Dyninst::ParseAPI::CodeObject::parse() [libp
	Dyninst::ParseAPI::Parse() [libparse
	Dyninst::ParseAPI::Parser::parse_vanilla() [
	Dyninst::ParseAPI::Parser::parse_frames(Lock
	[] Dyninst::ParseAPI::Parser::ProcessFrames
	Dyninst::InstructionAPT::Instruction.HInstruction(Dyninst::ComParallel []ihoom.so.1.0.0]
	uction const) [libinstructionAPI.so.12.0.1]
	11] LOCKFreeQueue <dyninst::parseapi::parsefr< td=""></dyninst::parseapi::parsefr<>
	[1] Inlined from Parser. C: 61/
	GMP_task [liggmp.so.i.0.0]
	Dyninst:::ParsePI:::ParsePI:::Duranterior
	Uninst :: Franser: Decord Options (D
	Dynins :: rial sear : rial s
	by hits trial sever transfer
	Uninst: ParseAPT: ParseAPT
	Dyninst::InspAdapter::IA x86::isTailCall(Dyn
	Dyninst::ParseAPI::StandardParseData::findBl
	[]] Dyninst::ParseAPI::region data::findBloc
	[I] Dyninst::dyn_c_hash_map <unsigned d<="" long,="" td=""></unsigned>
	[I] tbb::interface5::concurrent_hash_map <uns.< td=""></uns.<>
	Mini man
0s 2s 4s 6s 8s 10s 12s 14s 16s 18s 20s 22s 24s 26s 28s 30s 32s 34s	bs 385 405 425 44s 46s 48s 50s 52s 54s 56s 58s 60s 62s 64s 66s 68s 70s 72s 74s 76s
Depth view Summary view	Û+ Û-

Improved Tracing to Show Blocking on CPU Threads

Miniqmc: OpenMP on 32 CPU threads







Coarse- and Fine-grain Measurement on NVIDIA GPUs: LLNL's Quicksilver

Compute Node

- 2xPower9 + 4xNIVIDIA GPUs
- Optimized (-O2) compilation with nvcc
- Detailed measurement and attribution using PC sampling
- Attribute information to heterogeneous calling context
- Key Metrics
 - instructions executed
 - instruction stalls and reasons
 - GPU utilization

FIGILLE. 45					
relearData.cc %					
4 HOST DEVICE END					
5					
6// Return the total cross section for this energy group 7HOST DEVICE					
8 double NuclearData::getReactionCrossSection(
9 unsigned int reactIndex, unsigned int isotopeIndex, unsigned int group)					
<pre>u qs_assert(isotopeIndex < _isotopes.size());</pre>					
2 qs_assert(reactIndex < _isotopes[isotopeIndex]. species[0]. reactions.s:	ize());				
<pre>3 return _isotopes(isotopeindex)species(0)reactions(reactindex).gettro 4}</pre>	bssSection(group);				
5 HOST_DEVICE_END					_
7			GPU inst	GPU i	nst
G	- CPU tir	ne 📃 🚽			
p-down view Bottom-up view Flat view	0. 0		counts	stall	S
					_
11 🕂 🌀 🕅 💹 🕅 🧮 At 🗛 III 🕆 🖾					
Scone	EALTIME (sec) Sum (T) REAL	TIME (sec).Sum (E)	GTNS:Sum (T) GTNS:Sum (E)	TNS+STL ANY+Sum (T) GTN	IS+STL ANV+Sum
A <program root=""></program>	6 07e+00 100 0%	11HE (360).50m (E)	9 53e+10 100 0%	8 41e+10 100 0%	5.512_ART.50
∠ ⇒ main	6.07e+00 100.0%		9.53e+10 100.0%	8.41e+10 100.0%	
▲ loop at main.cc: 55	3.27e+00 53.9%		9.53e+10 100.0%	8.41e+10 100.0%	
✓ 58 → cvcleTracking(MonteCarlo*)	2.59e+00 42.7%	1.00e-02 0.	9.53e+10 100.0%	8.41e+10 100.0%	
✓ loop at main.cc: 159	2.59e+00 42.7%		9.53e+10 100.0%	8.41e+10 100.0%	
✓ loop at main.cc: 159	2.59e+00 42.7%		9.53e+10 100.0%	8.41e+10 100.0%	
▲ loop at main.cc: 163	2.57e+00 42.4%	1.00e-02 0.	s 9.53e+10 100.0%	8.41e+10 100.0%	
▲ 193 → [I] CycleTrackingKernel(MonteCarlo*, int, ParticleVault*.	2.38e+00 39.3%		9.53e+10 100.0%	8.41e+10 100.0%	
▲ 127 →device_stub_Z19CycleTrackingKernelP10MonteCarloiP13	. 2.38e+00 39.3%		9.53e+10 100.0%	8.41e+10 100.0%	
▲ 14 ⇒ [I] cudaLaunchKernel <char></char>	2.38e+00 39.3%		9.53e+10 100.0%	8.41e+10 100.0%	
✓ 211 → cudaLaunchKernel [qs]	2.38e+00 39.3%		9.53e+10 100.0%	8.41e+10 100.0%	
⊿ → <gpu kernel=""></gpu>			9.53e+10 100.0%	8.41e+10 100.0%	
▲ ⇒ CycleTrackingKernel(MonteCarlo*, int, ParticleVault*,.	•••		9.53e+10 100.0% 3.98e+07 0.0%	8.41e+10 100.0%	3.51e+07
✓ 132 → CycleTrackingGuts(MonteCarlo*, int, ParticleVau			9.52e+10 99.9% 5.17e+09 5.4%	8.41e+10 99.9%	5.15e+09
▲ 26 → [I] CycleTrackingFunction(MonteCarlo*, MC_Parti.	••		7.18e+10 75.4% 3.30e+08 0.3%	6.08e+10 72.3%	2.85e+08
Ioop at CycleTracking.cc: 118			7.18e+10 75.3% 2.99e+08 0.3%	6.08e+10 72.2%	2.59e+08
▲ 63 → CollisionEvent(MonteCarlo*, MC_Particle&, un.			4.60e+10 48.3% 4.12e+09 4.3%	3.83e+10 45.6%	3.20e+09
loop at CollisionEvent.cc: 67			3.65e+10 38.4% 7.32e+08 0.8%	2.99e+10 35.5%	5.73e+08
loop at CollisionEvent.cc: 71			3.45e+10 36.2% 2.30e+09 2.4%	2.82e+10 33.5%	1.69e+09
✓ 73 → macroscopicCrossSection(MonteCarlo*, int.	••		3.20e+10 33.6% 1.03e+10 10.8%	2.63e+10 31.3%	8.08e+09
			1 97e+10 10 7% 0 69e+00 10 2%	1 58e+10 18 8%	8.25e+09
▲ 41 → NuclearData::getReactionCrossSection(u			1.0/0410 19.7% 9.000409 10.2%	11500110 10100	

K. Zhou, M. W. Krentel, and J. Mellor-Crummey. Tools for top-down performance analysis of GPU-accelerated applications. International Conference on Supercomputing. ACM, New York, NY, USA, June, 2020.

Analysis of PeleC using PC Sampling on an NVIDIA GPU

hpoviewe Profile: PeleC3d.gnu.TPROF.CUDA.ex reactor.cop 💥 Metric properties 438 UserData udata = static_cast<ARKODEUserData*>(user_data); Cause: 439 udata->dt save = t; 440 441 #ifdef AMREX_USE_GPU passed udata structure pointer to lambda capture 442 const auto ec = amrex::Gpu::ExecutionConfig(udata->ncells d); 443 amrex::launch global<<< udata->nbBlocks, udata->nbThreads, ec.sharedMem, udata->stream>>>(444 [=] AMREX_GPU_DEVICE() noexcept { for (int icell = blockDim.x * blockIdx.x + threadIdx.x. Improvement: 447 stride = blockDim.x * gridDim.x; 448 icell < udata->ncells_d; icell += stride) { 449 fKernelSpec(450 icell, udata->dt save, udata->ireactor type, yvec d, ydot d, pass udata components as scalars 451 udata->rhoe init d. udata->rhoesrc ext d. udata->rYsrc d); 452 https://github.com/AMReX-Combustion/PelePhysics/pull/192 }); 453 454 #else 4% speedup on PeleC PMF drm19 test case 455 for (int icell = 0; icell < udata->ncells_d; icell++) { fKernelSnec(156 Top-down view Bottom-up view Flat view

🕆 🕀 💧 fo 🐖 🕅 🗚 🗛 💷 🕅

Scope		GINS.[0,0] (I)	•	GINS.[0,0] (E)		GINS:STL_ANY.[0,0]	(I)	GINS:STL_ANY.[0,0] (E)	GINS:STL_GMEM.[0,0]	(1)	GINS:STL_GMEM.[0,0]	(E)
Ioop at AMReX_Amr.cpp: 2061		1.24e+13	88.6%			1.05e+13	88.7%		5.58e+12	89.3%		
▲ ➡ 2062: amrex::Amr::timeStep(int, double, int, int, double)		1.24e+13	88.6%			1.05e+13	88.7%		5.58e+12	89.3%		
✓ ➡ 2015: PeleC::advance(double, double, int, int)	CPU	1.24e+13	88.5%			1.05e+13	88.6%		5.57e+12	89.2%		
# B 36: PeleC::do_sdc_advance(double, double, int, int)	0.0	1.24e+13	88.5%			1.05e+13	88.6%		5.57e+12	89.2%		
Ioop at Advance.cpp: 302	context	1.24e+13	88.4%			1.05e+13	88.5%		5.57e+12	89.1%		
S08: PeleC::do_sdc_iteration(double, double, int, int, int, int)	CONTOAL	1.24e+13	88.4%			1.05e+13	88.5%		5.57e+12	89.1%		
✓ ➡ 561: PeleC::react_state(double, double, bool, amrex::MultiFab*)		9.61e+12	68.5%			8.29e+12	70.0%		4.17e+12	66.8%		
loop at React.cpp: 109		9.43e+12	67.2%			8.14e+12	68.7%		4.06e+12	65.0%		
▲ B 210: react(amrex::Box const&, amrex::Array4 <double> const&, and a statement of the statement of the</double>	mrex::Array4 <double> cons</double>	9.39e+12	66.9%			8.10e+12	68.4%		4.03e+12	64.5%		
▲ ➡ 234: arkEvolve [libsundials_arkode.so.4.7.0]		9.28e+12	66.2%			8.00e+12	67.6%		3.94e+12	63.1%		
erkStep_TakeStep [libsundials_arkode.so.4.7.0]		7.16e+12	51.1%			6.19e+12	52.3%		3.05e+12	48.9%		
EF_RHS(double, _generic_N_Vector*, _generic_N_Vector*, void	*)	6.27e+12	44.7%			5.49e+12	46.3%		2.48e+12	39.7%		
▲ ➡ 443: [I] amrex::launch_global <nv_dl_wrapper_t<nv_dl_tag< p=""></nv_dl_wrapper_t<nv_dl_tag<>	<int (*)(double,="" _generic_n<="" th=""><th>6.27e+12</th><th>44.7%</th><th></th><th></th><th>5.49e+12</th><th>46.3%</th><th></th><th>2.48e+12</th><th>39.7%</th><th></th><th></th></int>	6.27e+12	44.7%			5.49e+12	46.3%		2.48e+12	39.7%		
I2: [I]wrapperdevice_stub_launch_global <nv_dl_wrap< p=""></nv_dl_wrap<>	per_t <nv_dl_tag<int (*)(do<="" th=""><th>6.27e+12</th><th>44.7%</th><th></th><th></th><th>5.49e+12</th><th>46.3%</th><th></th><th>2.48e+12</th><th>39.7%</th><th>mostly memo</th><th>orv</th></nv_dl_tag<int>	6.27e+12	44.7%			5.49e+12	46.3%		2.48e+12	39.7%	mostly memo	orv
4 # 26: [I]device_stub_ZN5amrex13launch_globallZ6cF_RF	ISdP17_generic_N_VectorS2	6.27e+12	44.7%			5.49e+12	46.3%		2.48e+12	39.7%	stalls	,
▲ 🖶 24: [I] cudaLaunchKernel <char></char>		6.27e+12	44.7%			5.49e+12	46.3%	9.4% GPU stalls	2.48e+12	39.7%	otano	
▲ Description A State A St		6.27e+12	44.7%			5.49e+12	46.3%	outside the loop	2.48e+12	39.7%		
GPI / B <gpu kernel=""></gpu>		6.27e+12	44.7%			5.49e+12	46.3%		2.48e+12	39.7%		
A B amrex::launch_global <cf_rhs(double, _generic_n_ve<="" th=""><th>ector*, _generic_N_Vector*, v</th><th>6.27e+12</th><th>44.7%</th><th>1.75e+10</th><th>0.1%</th><th>5.49e+12</th><th>46.3%</th><th>1.70e+10 0.1</th><th>% 2.48e+12</th><th>39.7%</th><th></th><th></th></cf_rhs(double,>	ector*, _generic_N_Vector*, v	6.27e+12	44.7%	1.75e+10	0.1%	5.49e+12	46.3%	1.70e+10 0.1	% 2.48e+12	39.7%		
Context 12: [I] cF_RHS(double, _generic_N_Vector*, _generic	_N_Vector*, void*)::{lambda	6.25e+12	44.6%	1.17e+12	8.3%	5.47e+12	46.2%	1.16e+12 9.8	% 2.4 8e+12	39.7%	1.14e+12	18.2%
► loop at reactor.cpp: 446		5.14e+12	36.6%	5.35e+10	0.4%	4.36e+12	36.8%	4.62e+10 0.4	% <u>1 38e+12</u>	<u>,</u> 22 08∉	3.29e+10	0.5%
reactor.cpp: 446		1.11e+12	7.9%	1.11e+12	7.9%	1.11e+12	9.4%	1.11e+12 9.4	% 1.10e+12	17.7%	1.10e+12	17.7%
AMReX_GpuLaunchGlobal.H: 12		1.75e+10	0.1%	1.75e+10	0.1%	1.70e+10	0.1%	1.70e+10 0.1	8			





HPCToolkit Trace of WarpX (16 ranks + 16 GPUs)

Time Range: [0s, 35s] Rank Range: [Rank 0 Thread 0, Rank 15 Thread 501] Cross Hair: (17s, Rank 8 Thread 0)





GPU imbalance: significantly less GPU work on last 6 ranks



Measure and Attribute OpenMP Offloading

hpcviewer		_			巴 [1]			
File View Filter Help								
편 Profile: miniqmc 🗮 Trace: miniqmc								
einspline_spo_omp.cpp 🛛								
<pre>160 auto* restrict psi_ptr = offload_scratch[i].dat</pre>	a();							
161 162 PRAGMA OFFLOAD("omp target teams distribute num teams(NumTeams) thread limit(Ch	unkSizePerTeam)")						
<pre>163 for (int team_id = 0; team_id < NumTeams; team_id++)</pre>								
<pre>104 1 165 const int first = ChunkSizePerTeam * team_id;</pre>								
<pre>166 const int last = 167 (first + ChunkSizePerTeam) > nSplinePerPlack la</pre>	cal 2 nSplingsPorPlack la	cal , first , Churd	SizoPorToom,					
168	cat : hsptimesrenblock_to	tat : TINST + Chunn	(Sizerei Tealli;					
169 int ix. iv. iz:								
Top-down view Bottom-up view Flat view								
111 - 🔀 👫 🔛 🖓 🖬 🗛 🗛 🖬 - 🖬								
Scope	GPUOP (sec):Sum (I)▼ GP	JOP (sec):Sum (E)	GKER (sec):Sum (I)	GKER (sec):Sum (E)	GMEM (sec):Sum (I)			
Experiment Aggregate Metrics	6.23e+00 100.0%	6.23e+00 100.0%	5.50e+00 100.0%	5.50e+00 100.0%	7.10e-03 100.0%			
<program root=""></program>	6.23e+00 100.0%		5.50e+00 100.0%	5	7.10e-03 100.0%			
⊿ → main	6.23e+00 100.0%		5.50e+00 100.0%	5	7.10e-03 100.0%			
▲ loop at miniqmc.cpp: 402	6.02e+00 96.6%		5.33e+00 96.9%	5				
▲ 404 → .omp_outlined62	6.02e+00 96.6%		5.33e+00 96.9%	5				
▲ 404 → [I] .omp_outlineddebug61	6.02e+00 96.6%		5.33e+00 96.9%	5				
▲ loop at miniqmc.cpp: 405	6.02e+00 96.6%		5.33e+00 96.9%	5				
▲ loop at miniqmc.cpp: 472	5.01e+00 80.5%		4.49e+00 81.6%	5				
▲ loop at miniqmc.cpp: 476	5.01e+00 80.5%		4.49e+00 81.6%	5				
▲ loop at miniqmc.cpp: 478	5.01e+00 80.5%		4.49e+00 81.6%	5				
▲ 485 ⇒ qmcplusplus::WaveFunction::ratio(q	5.01e+00 80.5%		4.49e+00 81.6%	5				
▲ ⇒ qmcplusplus::DiracDeterminant <qmcplus< p=""></qmcplus<>	5.01e+00 80.5%		4.49e+00 81.6%	5				
<pre>▲ 198 → qmcplusplus::einspline_spo_omp<</pre>	5.01e+00 80.5%		4.49e+00 81.6%	5				
▲ 187 → qmcplusplus::einspline_spo_omp	5.01e+00 80.5%		4.49e+00 81.6%	5				
loop at einspline_spo_omp.cpp: 157	5.01e+00 80.5%		4.49e+00 81.6%	5				
⊿ 162 → <omp kernel="" tgt=""></omp>	4.49e+00 72.0%		4.49e+00 81.6%	5				
⊿ → <gpu kernel=""></gpu>	4.49e+00 72.0%	4.49e+00 72.0%	4.49e+00 81.6%	4.49e+00 81.6%				
<pre>cunknown files []ibbnerun col. 0</pre>	4 400-00 72 08	4 400-00 72 09	4 400-00 91 69	1 100-00 91 69				







HPCToolkit Status on GPUs

- NVIDIA
 - heterogeneous profiles, including GPU instruction-level execution and stalls using PC sampling
 - traces
- AMD
 - heterogeneous profiles; no GPU instruction-level measurements within kernels
 - measure OpenMP offloading using OMPT interface
 - traces
- Intel
 - heterogeneous profiles, including GPU instruction-level measurements with kernel instrumentation and heuristic latency attribution to instructions
 - traces





Coming Attraction: Improved Scalability of Post-mortem Analysis

- Exploit natural sparsity in performance data
 - -Reduce storage requirements, efficiently use available I/O
- Use multithreading to process performance data
 - -Reduce memory footprint and communication cost, efficiently use available compute
- Empirical results of improvements in HPCToolkit
 - -Practical benefits: process data from 1000s of nodes with <10, in minutes!





Storing Mountains of Performance Data from Extreme-Scale Executions



Exploit natural sparsity to reduce storage and I/O

"1254x compression: 14TB→11GB for PeleC (turbulent combustion) @ 2K threads + 2K GPUs"





Analyzing Mountains of Performance Data from Extreme-Scale Executions



Highly-efficient multithreaded parallelism!





Scalable parallelism: multithreading + MPI



hpcprof-mpi: Analyze Measurements of LAMMPS @ 2K threads + 2K GPUs



Analysis on 8 nodes using 504 threads!

HPCToolkit Resources

- Documentation
 - User manual
 - <u>http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf</u>
 - Tutorial videos
 - <u>http://hpctoolkit.org/training.html</u>
- Software
 - Download hpcviewer GUI binaries for your laptop, desktop, cluster, or supercomputer
 - OS: Linux, Windows, MacOS
 - Processors: x86_64, aarch64, ppc64le
 - <u>http://hpctoolkit.org/download.html</u>
 - Install HPCToolkit on your Linux desktop, cluster, or supercomputer using Spack
 - <u>http://hpctoolkit.org/software-instructions.html</u>







HPCToolkit Hands-On Directions

Performance analysis of CPU and GPU-accelerated applications

John Mellor-Crummey Professor, Rice University



Sample Performance Databases for You to Explore

- Where can you find the databases: theta:/grand/ATPESC2022/hpctoolkit/data
 - -hpctoolkit-gamess.makefp.crusher.db
 - General Atomic and Molecular Electronic Structure System (GAMESS) is a general ab initio quantum chemistry package
 - Fortran; MPI + OpenMP offloading (Cray CCE); AMD GPUs
 - 110s; 40MB; 16 MPI ranks x (5 CPU threads + 2 GPU streams)
 - -hpctoolkit-qmcpack-database-dmc-S16-cpu-n32-t32-d2-BGQ
 - An early prototype distributed-memory implementation of QMCPACK a many-body ab initio Quantum Monte Carlo code for computing the electronic structure of atoms, molecules, 2D nanomaterials & solids
 - C++; MPI + OpenMP; Blue Gene Q
 - 155s; 3.2GB; 32 MPI ranks x 32 threads
 - -hpctoolkit-PeleC-PMF-96GPU.d
 - PeleC is an adaptive-mesh compressible hydrodynamics code for reacting flows
 - C++; AMReX framework using CUDA ; Power9 + NVIDIA GPUs
 - 1.4GB; 96 MPI ranks x (3 threads + 5 GPU streams)
 - -hpctoolkit-PeleC3d.dpcpp.ex-skylake-gpu.d
 - C++; AMReX framework using SYCL ; Intel Skylake with integrated GPU cores
 - Instruction-level measurements within GPU kernels
 - 10s; 44MB; Single process + GPU offloading





Profiling Quicksilver with HPCToolkit on Theta-gpu

- module swap cobalt/cobalt-knl cobalt/cobalt-gpu
- # if cobalt/cobalt-gpu is loaded

- ssh thetagpusn1
- qsub -I -q single-gpu -t 60 -n 1 --attrs filesystems=grand -A ATPESC2022
- source /grand/ATPESC2022/hpctoolkit/scripts/setup-proxy.sh
- cd /grand/ATPESC2022/usr/\${LOGNAME}
- git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
- cd hpctoolkit-tutorial-examples/examples/gpu/quicksilver
- source setup-env/theta-gpu.sh
- make build
- make run
- make run-pc
- exit # your compute node
- exit # thetagpusn1
- cd /grand/ATPESC2022/usr/\${LOGNAME}
- cd hpctoolkit-tutorial-examples/examples/gpu/quicksilver
- module load hpctoolkit
- hpcviewer hpctoolkit-qs-gpu-cuda.d
- hpcviewer hpctoolkit-qs-gpu-cuda-pc.d





Profiling AMG2013 with HPCToolkit on Theta

- module swap cobalt/cobalt-gpu cobalt/cobalt-knl # if cobalt/cobalt-gpu is loaded
- cd /grand/ATPESC2022/usr/\${LOGNAME}
- git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
- cd hpctoolkit-tutorial-examples/examples/cpu/mpi+openmp/amg2013
- export HPCTOOLKIT_TUTORIAL_RESERVATION=<queue name>
- export HPCTOOLKIT_TUTORIAL_PROJECTID=ATPESC2022
- source setup-env/theta.sh
- make build
- make run
- # wait for \$COBALT_JOBID.done to appear in your directory
- Alternatives
 - make analyze
 - make analyze-parallel
- # wait for \$COBALT_JOBID.done to appear in your directory
- Alternatives
 - make view
 - hpcviewer hpctoolkit-amg2013.d



