

Efficiently Representing CPU-GPU Performance

Jonathon Anderson Rice University

July 7, 2025





- Novel enhancements to performance data representation
 - Same data, but in a smaller size
- Adjustments to improve parallelism in post-mortem analysis
 - Same high-level structure, but written faster
- Ongoing work

Examples: Calling Context Tree (CCT)





Image credit: LLNL/Hatchet

Examples: Flat Sampling / Flat Vector



# Address	Source	Live Registers no	_sample_count	Sampling Data (All)	Sampling ^ (Not Issi
22 00007f72 42fa6a50	IADD3 R5, R3, 0x1 , RZ	5	40	40	Contraction Control
23 00007f72 42fa6a60	MOV R28, 8x8		40	40	
24 00007f72 42fa6a70	IMAD R21, R3, R2, R0		16	16	
25 00007f72 42fa6a80	IMAD R5, R2, R5, R0		32	32	
26 00007f72 42fa6a90	IMAD R3, R3, R2, -R2		34	34	
27 00007f72 42fa6aa0	IMAD.WIDE R16, R21, R20, c[0x0][0x1a0]		32	32	-
28 00007f72 42fa6ab0	IADD3 R3, R0, R3, RZ		46	46	
29 00007f72 42fa6ac0	IMAD.WIDE R10, R5, R20, c[0x0][0x1a0]		19	19	
30 00007f72 42fa6ad0	IMAD.WIDE R18, R3, R20, c[0x0][0x1a8]	— 11	83	83	
31 00007f72 42fa6ae0	LDG.E.64.CONSTANT.SYS R16, [R16]	1 0	44	44	
32 00007f72 42fa6af0	IMAD.WIDE R26, R21, R28, c[8x8][8x198]	12	44	44	
33 00007f72 42fa6b00	LDG.E.64.CONSTANT.SYS R10, [R10]	12	41	41	
34 00007f72 42fa6b10	IMAD.WIDE R28, R21, R26, c[8x0][0x1a8]	14	38	38	
35 00007f72 42fa6b20	LDG.E.64.CONSTANT.SYS R18, [R18]	14	41	41	
36 00007f72 42fa6b30	IMAD.WIDE R12, R5, R28, c[8x8][8x1a8]	16	52	52	
37 00007f72 42fa6b40	LDG.E.64.CONSTANT.SYS R14, [R26]	17	42	42	
38 00007f72 42fa6b50	LDG.E.64.CONSTANT.SYS R6, [R26+8x8]	19	100	100	
39 00007f72 42fa6b60	LDG.E.64.CONSTANT.SYS R2, [R28+-0x8]	19	137	137	
40 00007f72 42fa6b70	LDG.E.64.CONSTANT.SYS R12, [R12]	19	180	180	
41 00007f72 42fa6b80	LDG.E.64.CONSTANT.SYS R8, [R28+0x8]	21	115	115	
42 00007f72 42fa6b90	LDG.E.84.CONSTANT.SYS R4, [R28]	23	99	99	
43 00007f72 42fa6ba0	IMAD.WIDE R20, R21, R20, c[0x0][0x190]	21	20	20	
44 00007f72 42fa6bb0	LDG.E.64.SYS R22, [R20]	23	177	177	
45 00007f72 42fa6bc0	DADD R24, R16, R10	25	22,607	22,607	21
46 00007f72 42fa6bd0	DMUL R18, R16, R18	25	1,82	1,821	1,
47 00007f72 42fa6be0	DADD R24, R24, 1	23	2,683	2,683	2,
48 00007f72 42fa6bf0	DADD R16, R14, R6	25	1,32 0	1,320	1,
49 00007f72 42fa6c00	DMUL R2, R14, R2	25	1,338	1,338	1,
50 00007f72 42fa6c10	DADD R16, R24, R16	23	1,279	1,279	1,
51 00007f72 42fa6c20	DFMA R12, R10, R12, R18	21	427	427	
52 00007f72 42fa6c30	DFMA R2, R6, R8, R2	17	423	423	
53 00007f72 42fa6c40	DFMA R4, R16, R4, -R12	13	2,15	2,151	1

••••••

Image credit: Nvidia

DVIDIA

Examples: Call Graph





HPCToolkit's Instruction-Level Attribution within GPU Kernels (CCT)



le View Filter Help							
Profile: qs							-
in.cc CollisionEvent.cc ×							
<pre>int uniqueNumber = monteCarlo->_materialDatabase->_mat[globalMatIndex]iso[i:</pre>	soIndex]gid;						
<pre>int numReacts = monteCarlo->_nuclearData->getNumberReactions(uniqueNumber);</pre>							
<pre>for (int reactIndex = 0; reactIndex < numReacts; reactIndex++) f</pre>							
currentCrossSection -= macroscopicCrossSection(monteCarlo, reactIndex, mc_	particle.domain, mc_particle.cell,						
isoIndex, mc_particle.energy_group); if (currentCrossSection < 0)							
{							
<pre>selectedIso = isoIndex;</pre>							
selectedUniqueNumber = uniqueNumber; selectedReact = reactIndex;							
break;							
)							
)							
qs_assert(selectedIso != -1);							
-down view Bottom-up view Flat view							
N 11 4 10 101 101 101 10 10 101							
🙀 🔸 🕼 🖾 🖾 🗛 🗛 👘 👘 👘							
	GTNS: Sum (T) = GTNS: Sum (E)	GTNS-STL ANY- Sum (T)	GTNS-STL ANY: SUM (E)	GTNS-STI TEET- Sum (T)	GING STI TEET Sug		T1 T
cope	GINS: Sum (I) ▼ GINS: Sum (E)	GINS:STL_ANY: Sum (I) 1.190+11 100.0		GINS:STL_IFET: Sum (I) 5.2/0+09 100.0%	GINS:STL_IFET: Sum	(E) GINS:S	
<pre><pre></pre> ▲ 14 ⇒ [1] cudaLaunchkernel<char></char></pre>			16		GINS:STL_IFET: Sum	(E) GINS:S	9
cope ▲ 14 ≫ [1] cudaLaunchkernel <char> ▲ 211 ≫ cudaLaunchKernel [qs]</char>	1.390+11 100.0%	1.190+11 100.0	n	5.2/0+09 100.0%	GINS:STL_IFET: Sum	(E) GINS:S	9
ope ▲ 14 » [1] cudaLaunchKernel <char> ▲ 211 » cudaLaunchKernel [qs] ▲ » <gpu kernel=""></gpu></char>	1.30e+11 100.0% 1.30e+11 100.0%	1.19e+11 100.0 1.19e+11 100.0 1.19e+11 100.0	n — — — — — — — — — — — — — — — — — — —	5.27e+89 180.0% 5.27e+89 180.0% 5.27e+89 180.0%	_	0.4%	9 9 9
<pre>cope 14 > [1] cudaLaunchkernel<cnar> 211 > (udaLaunchkernel [qs]</cnar></pre>	1.39e+11 109.0% 1.39e+11 100.0% 1.39e+11 109.0%	1.19e+11 100.0 1.19e+11 100.0 1.19e+11 100.0 3% 1.19e+11 100.0	ი ი ი ი ა 3.62e+07 0.0%	5.270+89 100.0% 5.270+89 100.0% 5.270+89 100.0% 5.270+89 100.0%	_		9 9 9 9
<pre>cope 4.14 » [1] cudaLaunchKernel<cnar- 4="" 4.132="" 4.211="" [qs]="" cudalaunchkernel="" cycleirackingkernel(montecarlo*,="" cycleirackingkuts(montecarlo*,="" int,="" kernel="" particlevau="" particlevault*,="" particlevault*,<="" sgpu="" td="" »=""><td>1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 4.08c+07 0.0</td><td>1.190+11 100.0 1.190+11 100.0 1.190+11 100.0 3% 1.190+11 1% 1.190+11</td><td>ო – რ რ რ 3.62c+07 0.0% რ 9.01c+09 7.6%</td><td>5.2/0+09 100.0% 5.270+09 100.0% 5.270+09 100.0% 5.270+09 100.0% 5.270+09 100.0% 5.240+09 99.5%</td><td>- 2.11c+07</td><td>0.4%</td><td>9 9 9 9</td></cnar-></pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 4.08c+07 0.0	1.190+11 100.0 1.190+11 100.0 1.190+11 100.0 3% 1.190+11 1% 1.190+11	ო – რ რ რ 3.62c+07 0.0% რ 9.01c+09 7.6%	5.2/0+09 100.0% 5.270+09 100.0% 5.270+09 100.0% 5.270+09 100.0% 5.270+09 100.0% 5.240+09 99.5%	- 2.11c+07	0.4%	9 9 9 9
opo 214 % [1] cudaLaunchKernel (char> 211 » cudaLaunchKernel [45] 4 » cgot kernel> 4 » cgotelrackingKernel/MonteCarlo*, int, ParticleVault*, ParticleVau. 4 132 » CycleTrackingKuts(MonteCarlo*, int, ParticleVault*, ParticleV. 4 28 » [1] cycleTrackingFunction(MonteCarlo*, MC_ParticleS, int, P., .	1.30e+11 100.0% 1.30e+11 100.0% 1.30e+11 100.0% 1.30e+11 100.0% 4.08e+07 0.0 1.30e+11 100.0% 9.03e+09 7.00	1.19e+11 160.0 1.19e+11 100.0 1.19e+11 100.0 3% 1.19e+11 1.19e+11 100.0 3% 1.19e+11 3% 1.19e+11 3% 7.25e+10	% % % % 3.62c+07 0.0% % 9.01c+09 7.6% % 3.65c+08 0.3%	5.2/6+09 100.0% 5.27c+09 100.0% 5.27c+09 100.0% 5.27c+09 100.0% 5.27c+09 100.0% 5.24c+09 99.5% 5.21c+09 98.9%	- 2.11e+07 8.98e+06	0.4%	9 9 9 9 9
<pre>ope</pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 4.080+07 0.1 1.300+11 100.0% 9.030+09 7. 8.360+10 04.4% 4.120+08 0.3 8.350+10 04.4% 3.760+08 0.3 8.350+10 04.4% 3.760+08 0.3	I.19e+11 169.8 I.19e+11 160.8	%	5.2/6+89 189.0% 5.276+89 180.0% 5.276+89 180.0% 5.276+89 180.0% 5.246+89 99.5% 5.240+89 98.9% 5.210+89 98.8%	- 2.11e+07 8.98e+06 1.02e+08	0.4% 0.2% 1.9%	9 9 9 9 9 9
<pre>ope</pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 4.080+07 0.1 1.300+11 100.0% 9.030+09 7. 8.360+10 04.4% 4.120+08 0.3 8.350+10 04.4% 3.760+08 0.3 8.350+10 04.4% 3.760+08 0.3	1.19e+11 108.6 1.19e+11 108.6 1.19e+11 108.6 3% 1.19e+11 1.9e+11 108.6 3% 7.25e+10 3% 7.25e+10 3% 4.44e+10	%	5.276499 100.0% 5.276499 100.0% 5.276499 100.0% 5.276499 100.0% 5.246499 99.5% 5.216499 98.8% 5.216499 98.8% 3.056409 73.1%	2.11e+07 8.98e+06 1.02e+08 9.90e+07	0.4% 0.2% 1.9% 1.9%	9 9 9 9 9 9 9 6
<pre>ope 14 * [1] cudaLaunchKernel<char> 211 * cudaLaunchKernel [gs] 4 * cgpu kernel 4 * CycleIrackingKernel(MonteCarlo*, int, ParticleVault*, Partivaut*, ParticleVault*, ParticleVaut*, ParticleVaut*, Parti</char></pre>	1.30+11 100.0% 1.30+11 100.0% 1.30+11 100.0% 1.30+11 100.0% 1.30+11 100.0% 8.36+10 64.3% 8.35+10 64.3% 5.20+10 40.1% 4.99+09 3.0	1.196+11 108.8 1.196+11 108.8 1.196+11 108.8 3% 1.196+11 3% 1.196+11 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10 3% 7.256+10	n	5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.244+09 99.5% 5.214+09 98.0% 5.214+09 98.0% 3.854+09 73.1%	2.11c+07 8.98c+96 1.02c+98 9.90c+07 4.89c+98 1.27c+98	0.4% 0.2% 1.9% 1.9% 9.3%	9 9 9 9 9 9 9 6 5
<pre>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>></pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 9.030+09 7.6 8.300+10 04.4% 4.120+08 0.1 8.350+10 04.3% 3.760+08 0.1 5.200+10 40.1% 4.990+09 0.1 4.090+10 31.5% 8.150+08 0.6	1.19e+11 100.0 1.19e+11 100.0 1.19e+11 100.0 3% 1.19e+11 1.9e+11 100.0 3% 7.25e+10 3% 7.25e+10 3% 7.25e+10 3% 7.25e+20 3% 7.25e+10 3% 7.25e+10 3% 3.42e+10 3% 3.42e+10 3% 3.42e+10	 S. 1.62c+07 O.01c+09 O.01c+09 O.05c+08 O.33x O.34c+08 O.34x O.2c+09 O.34x O.34c+08 O.34x O.34c+08 O.34x O.34c+08 O.34x O.34c+08 O.34x O.34c+08 O.34x O.34c+08 O.34x <lio.34x< li=""> O.34x <lio< td=""><td>5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.244+09 99.5% 5.214+09 99.5% 3.854+09 95.1% 3.854+09 97.1% 3.274+09 62.0%</td><td>2.11c+07 8.98c+96 1.02c+98 9.90c+07 4.89c+98 1.27c+98</td><td>0.4% 0.2% 1.9% 1.9% 9.3% 2.4% 4.3%</td><td>9 9 9 9 9 9 9 6 5 5</td></lio<></lio.34x<>	5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.244+09 99.5% 5.214+09 99.5% 3.854+09 95.1% 3.854+09 97.1% 3.274+09 62.0%	2.11c+07 8.98c+96 1.02c+98 9.90c+07 4.89c+98 1.27c+98	0.4% 0.2% 1.9% 1.9% 9.3% 2.4% 4.3%	9 9 9 9 9 9 9 6 5 5
<pre>ope 14 * [1] cudaLaunchKernel<char> 211 * cudaLaunchKernel [gs] 4 * cgpu kernel 4 * CycleIrackingKernel(MonteCarlo*, int, ParticleVault*, Partivaut*, ParticleVault*, ParticleVaut*, ParticleVaut*, Parti</char></pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 4.080+07 0.1 8.300+11 00.0% 8.350+10 64.3% 3.750+08 0.1 5.200+10 40.1% 4.990+09 3.1 3.050+10 2.0% 2.700+09 2.1	1.19+11.100.0 1.19+11.100.0 1.19+11.100.0 1.19+11.100.0 30 1.9+11.100.0 31 7.25+10 32 4.44+10 35 3.42+10 3.42+10 3.22+10 4.34+10 3.42+10 3.22+10 4.44+10 3.22+10 3.22+10 4.44+10	 G G	5.27449 100.0% 5.27449 100.0% 5.27449 100.0% 5.27449 100.0% 5.2449 99.5% 5.21449 99.5% 5.21449 99.5% 5.21449 99.8% 3.354409 73.1% 3.54409 77.1% 3.24409 77.1% 3.24409 57.1%	2.11e+87 8.98e+86 1.02e+88 9.98e+87 4.89e+88 1.27e+88 2.28e+08 1.79e+09	0.4% 0.2% 1.9% 1.9% 9.3% 2.4% 4.3%	9 9 9 9 9 9 6 5 5 4
<pre>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>></pre>	1.30+11 100.0% 1.30+11 100.0% 1.30+11 100.0% 1.30+11 100.0% 1.30+11 100.0% 0.03+00 7.0 8.36+10 64.4% 4.12e+08 0.1 8.350+10 64.4% 4.12e+08 0.1 8.350+10 0.1% 4.392+09 3.1 4.09+10 31.5% 8.15e+08 0.0 3.55e+10 29.6% 2.70e+09 9.2	1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 101.0 1.19+11 101.0 1.19+11 101.0 1.19+11 101.0 1.19+11 101.0 1.19+11 101.0 1.19+11 101.0	- - - -	5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.244-09 99.5% 5.214-09 98.8% 3.054-09 98.8% 3.354-09 67.1% 3.274-09 62.0% 3.044-09 57.7%	2.11c+87 8.98c+86 1.92c+98 9.98c+97 4.89e+198 1.27e+08 2.28e+198 2.28e+198 6.68e+198	0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9%	9 9 9 9 9 9 6 5 5 4 2
<pre>cope 14 * [1] CUGALSUNCHKETMEL<char- *="" **="" 211="" 22="" 4132="" [1]="" at="" cugalsunchketmel<[g]="" cycletracking-cucrils(montecarlo*,="" cycletracking-cucrils<="" cycletrackingguts(montecarlo*,="" cycletrackingkernel[g]="" cycletrackingkernel[montecarlo*,="" int,="" loop="" mc_particles,="" p="" particle="" particlevau="" particlevault*,="" td=""><td>1.30e+11 100.0% 1.30e+11 100.0% 1.30e+11 100.0% 1.30e+11 100.0% 3.30e+11 100.0% 3.30e+11 0.0% 3.30e+11 0.0% 3.30e+10 0.4.0% 4.12e+08 0.3 5.20e+10 0.4.1% 4.09e+10 1.5% 3.58e+10 2.70e+09 3.58e+10 2.75% 3.58e+10 2.75% 3.09e+10 1.6%</td><td>1.19e+11 100.0 1.19e+11 100.0 1.19e+11 100.0 3% 1.19e+11 1.9 1.9e+11 3% 1.9e+11 1.9 1.9e+11 3% 7.25e+10 3% 7.25e+10 3% 4.44e+10 3% 3.42e+10 3% 3.22e+10 4% 3.01e+10 4% 3.01e+10 3% 3.62e+10 4% 3.01e+10 4% 3.01e+10 4% 3.01e+10</td><td>- - - -</td><td>5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.224+09 99.5% 5.214+09 99.5% 3.654+09 97.5% 3.654+09 97.5% 3.274+09 62.0% 3.274+09 62.0% 1.064+09 25.8% 2.254+08 4.33</td><td>2.11c+87 8.98c+86 1.92c+98 9.98c+97 4.89e+198 1.27e+08 2.28e+198 2.28e+198 6.68e+198</td><td>0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9%</td><td>9 9 9 9 9 9 6 5 5 4 2 8</td></char-></pre>	1.30e+11 100.0% 1.30e+11 100.0% 1.30e+11 100.0% 1.30e+11 100.0% 3.30e+11 100.0% 3.30e+11 0.0% 3.30e+11 0.0% 3.30e+10 0.4.0% 4.12e+08 0.3 5.20e+10 0.4.1% 4.09e+10 1.5% 3.58e+10 2.70e+09 3.58e+10 2.75% 3.58e+10 2.75% 3.09e+10 1.6%	1.19e+11 100.0 1.19e+11 100.0 1.19e+11 100.0 3% 1.19e+11 1.9 1.9e+11 3% 1.9e+11 1.9 1.9e+11 3% 7.25e+10 3% 7.25e+10 3% 4.44e+10 3% 3.42e+10 3% 3.22e+10 4% 3.01e+10 4% 3.01e+10 3% 3.62e+10 4% 3.01e+10 4% 3.01e+10 4% 3.01e+10	- - - -	5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.224+09 99.5% 5.214+09 99.5% 3.654+09 97.5% 3.654+09 97.5% 3.274+09 62.0% 3.274+09 62.0% 1.064+09 25.8% 2.254+08 4.33	2.11c+87 8.98c+86 1.92c+98 9.98c+97 4.89e+198 1.27e+08 2.28e+198 2.28e+198 6.68e+198	0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9%	9 9 9 9 9 9 6 5 5 4 2 8
<pre>cope 14 > [1] cudaLaunchKern@l<char> 211 = cudaLaunchKern@l<char></char></char></pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 3.300+11 100.0% 3.300+10 00.4% 3.300+10 00.4% 3.750+10 40.1% 4.990+00 31.5% 3.500+10 20.6% 2.700+00 2.1% 3.5500+10 27.5% 1.220+10 00.4% 2.090+10 10.1% 3.590+10 27.5% 1.220+10 00.4% 3.990+10 10.1% 3.990+10 0.1% 3.770+00 2.5% 3.770+00 3.7% 3.770+00 3.7	1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196 1.196+11.100.0 1.196 1.196 1.196+11.100.0 1.196 </td <td>- - - -</td> <td>5.27449 100.0% 5.27449 100.0% 5.27449 100.0% 5.27449 100.0% 5.27449 99.5% 5.21449 99.5% 5.21449 99.5% 5.21449 99.5% 3.354419 07.1% 3.34419 07.1% 3.27499 62.0% 3.044109 57.7% 1.26419 57.7% 1.26419 57.4% 4.764108 9.0%</td> <td>2.11c+07 8.98c+06 1.92c+07 4.99c+08 1.27c+08 2.28c+08 1.79c+09 6.68c+08 8.24c+08 8.24c+08</td> <td>0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9% 12.7% 1.6%</td> <td>TL_I 9 9 9 9 9 9 9 9 9 9 5 5 5 5 5 5 5 5 5</td>	- - - -	5.27449 100.0% 5.27449 100.0% 5.27449 100.0% 5.27449 100.0% 5.27449 99.5% 5.21449 99.5% 5.21449 99.5% 5.21449 99.5% 3.354419 07.1% 3.34419 07.1% 3.27499 62.0% 3.044109 57.7% 1.26419 57.7% 1.26419 57.4% 4.764108 9.0%	2.11c+07 8.98c+06 1.92c+07 4.99c+08 1.27c+08 2.28c+08 1.79c+09 6.68c+08 8.24c+08 8.24c+08	0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9% 12.7% 1.6%	TL_I 9 9 9 9 9 9 9 9 9 9 5 5 5 5 5 5 5 5 5
<pre>cope</pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 3.350+10 64.3% 3.500+10 0.1% 3.500+10 0.1% 3.500+10 0.1% 3.500+10 2.700+10 3.500+10 2.75 3.500+10 2.75 3.090+10 1.100+10 6.000+00 5.3% 3.770+00 2.5 6.280+09 4.4%	1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 30 1.19+11 1.99 1.19+11 1.90 1.19+11 30 1.29+11 31 1.29+11 32 7.25+10 33 4.44+10 34 4.44+10 35 3.22+10 36 3.22+10 37 3.22+10 38 3.01+10 37 5.66+10 38 5.66+10 39 5.66+10 30 5.66+10 31 1.64+10	- - - -	5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.224-09 99.5% 5.214-09 98.6% 3.054-09 97.1% 3.544-09 67.1% 3.274-09 62.0% 3.274-09 62.0% 3.264-09 71.5% 4.3% 4.764-08 9.5% 0.124+07 1.5%	2.11c+07 8.98c+06 1.92c+07 4.99c+08 1.27c+08 2.28c+08 1.79c+09 6.68c+08 8.24c+08 8.24c+08	0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9% 12.7% 1.6% 9.0%	9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
<pre>cope</pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 3.300+11 00.0% 8.300+10 0.4.0% 8.300+10 0.4.0% 4.120+00 0.1 8.350+10 0.4.0% 4.090+10 1.5.4% 1.590+10 1.54% 1.590+10 1.54% 0.990+10 5.1% 1.0590+10 4.2% 1.850+00 4.2% 1.850+00 4.2%	1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 3% 1.19+11 1.19+11 100.0 3% 1.25+10 3% 7.25+10 3% 7.25+10 3% 3.42+10 3% 3.42+10 3% 3.22+10 3% 3.01+10 3% 3.62+10 4% 3.79+10 3% 5.66+09 4% 5.66+04 4% 1.64+00 4% 1.64+00 4% 1.64+00	- - - -	5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.274+09 100.0% 5.224+09 99.5% 5.214+09 99.5% 5.214+09 99.5% 3.55409 67.1% 3.55409 67.1% 3.274+09 62.0% 3.046+09 52.5% 2.254+08 4.3% 4.764+08 9.0% 1.106+08 2.1%	2.11e+97 8.98e+96 1.02e+98 9.90e+97 4.199+98 1.27e+08 1.27e+08 1.79e+09 6.66e+08 6.24e+07 4.75e+09 8.12e+07	0.4% 0.2% 1.9% 9.3% 2.4% 4.3% 33.9% 12.7% 1.6% 9.0% 1.5%	9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
<pre>cope 11 * 0 [1] cudaLaunchKernel<cnar> 211 * cudaLaunchKernel(gs]</cnar></pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 3.300+11 100.0% 3.300+10 04.4% 4.120+08 0.3 3.500+10 04.4% 4.120+08 0.3 3.750+10 40.1% 4.990+10 31.5% 3.1500+10 27.5% 1.220+10 04.2% 2.090+10 10.1% 3.0500+10 27.5% 1.220+10 04.2% 3.0500+10 5.3% 3.770+09 2.5 6.200+09 4.0% 6.200+09 4.0% 6.200+09 1.4% 1.600+09 1.2% 1.610+09 1.2%	1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 1.196+11.100.0 30 1.196+11.100.0 31.196+11.100.0 32 34.196+11.100.0 35.196+11.100.0 36.196+11.100.0 37.256+10.61.1 38.444+10.37.4 39.4.444+10.37.4 39.4.444+10.37.4 39.4.444+10.37.4 39.4.444+10.37.4 39.4.444+10.37.4 39.4.444+10.37.4 39.4.444+10.37.4 39.5.666+09.4.9 39.5.666+09.4.9 39.5.666+09.4.9 39.5.666+09.4.4 39.5.666+09.4.4 39.5.666+09.4.3 39.5.666+09.4.3 39.5.666+09.4.3 39.5.666+09.4.3 39.1.146+09.1.0	- - - -	5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.274-09 98.0% 5.214-09 98.0% 5.214-09 98.0% 3.059-09 73.1% 3.544-09 67.1% 3.274-09 62.0% 3.064-09 57.7% 1.264-09 23.6% 4.764-08 9.0% 0.124-07 1.5% 1.104-08 2.1% 7.374-04 0.0%	2.11c+97 8.98c+96 9.96c+97 4.09c+98 1.27c+98 2.28c+08 1.79c+09 6.058c+06 8.24c+07 4.76c+98 8.12c+77 1.10c+08	0.4% 0.2% 1.9% 2.4% 4.3% 33.9% 12.7% 1.6% 9.0% 1.5% 2.1% 0.0%	9. 9. 9. 9. 9. 9. 9. 9. 9. 9. 9. 9. 9. 9
<pre>cope</pre>	1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 100.0% 1.300+11 00.0% 1.300+11 00.0% 1.300+11 00.0% 1.300+11 00.0% 1.300+11 00.0% 1.500+10 04.1% 1.500+10 01.1% 1.500+10 01.1% 1.500+10 0.1% 1.500+10 0.1% 1.500+10 0.1% 1.610+00 1.4% 1.610+00 1.4% 1.610+00 1.4% 1.610+00 1.4%	1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 1.19+11 100.0 3% 1.19+11 1.19+11 100.0 3% 1.19+11 1.9 1.19+11 1.9 1.25+10 3% 7.25+10 3% 4.44+10 3% 3.42+10 3% 3.22+10 4% 3.22+10 5% 5.66+09 4% 5.66+09 3% 5.66+09 1.18+09 1.0 3% 1.14+09 3% 1.44+09 3% 9.40+00	- - - -	5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.274-09 100.0% 5.274-09 98.0% 5.214-09 98.0% 3.054-09 67.1% 3.274-09 62.0% 3.274-09 62.0% 3.274-09 62.0% 3.274-09 63.0% 3.274-09 63.0% 3.2254-00 23.0% 3.2254-00 23.0% 3.2254-00 23.0% 3.2254-00 23.0% 3.244-07 1.5% 1.106+08 2.1% 3.124+07 2.5% 3.442+05 0.0%	2.11e+97 8.98+96 1.02e+08 9.99e+97 4.999+98 2.28+98 1.79+98 6.68+98 8.24e+07 4.76e+98 6.12e+07 1.10e+06 7.37e+04	0.4% 0.2% 1.9% 2.4% 4.3% 33.9% 12.7% 1.6% 9.0% 1.5% 2.1% 0.0%	9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

HPCToolkit's Instruction-Level Attribution within GPU Kernels (CCT)



Scope					
<pre>4 14 » [1] cudaLaunchKernel<char></char></pre>					
✓ 211 » cudaLaunchKernel [qs]					
⊿ » <gpu kernel=""></gpu>					
SycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVau					
I32 » CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, Particle					
▲ 26 » [I] CycleTrackingFunction(MonteCarlo*, MC_Particle&, int, P					
▲ loop at CycleTracking.cc: 118					
▲ 63 ≫ CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int) [
▲ loop at CollisionEvent.cc: 67					
Ioop at CollisionEvent.cc: 71					
▲ 73 ≫ macroscopicCrossSection(MonteCarlo*, int, int, i					
✓ 41 » NuclearData::getReactionCrossSection(unsigned int, u					
253 » [I] NuclearDataReaction::getCrossSection(unsigned					
NuclearData.cc: 253					
NuclearData.cc: 251					
NuclearData.cc: 248					
<pre>> 252 >> [I] qs_vector<nucleardataspecies>::operator[](int)</nucleardataspecies></pre>					
NuclearData.cc: 252					
<pre>> 252 >> [I] qs_vector<nucleardatareaction>::size() const</nucleardatareaction></pre>					
• 252					

HPCToolkit's Measurement Approach



- HPCToolkit uses different representations (for measurement) depending on the performance measured
 - CPU performance: attribute samples to full calling context (CCT)
 - Application stack is unwound at runtime
 - GPU performance: attribute flat samples to instructions within each GPU kernel
 - Limitation of the GPU vendor runtime APIs
 - AMD GPU PC samples: correlation ids enable mapping to full CPU calling context of kernel launch
 - Nvidia & Intel GPU PC samples: can't be correlated to CPU calling context without a performance hit
 - GPU APIs don't provide correlation id; correlation accomplished by serializing all GPU kernels
- HPCToolkit's post-mortem analysis converts each of the representations to CCT

HPCToolkit's CCT-based Attribution



- Flat samples are apportioned across plausible call paths
 - Attributed to static call graph, then expanded to CCT
 - If a node has multiple callers (red), it's cost is distributed based on call counts
 - CCT contains multiple nodes for each GPU function, one for each plausible caller
 - ...Recursively to cover all plausible call paths



Call graph for one GPU kernel of Quicksilver (w/o optimization)

Shortcomings of CCT-based Attribution



- Mercury: Monte Carlo transport app developed at LLNL
 - ~2K functions in one GPU kernel
 - ~500K call paths to a single, widely-used leaf function
 - One function called cuda_div_... >100 times
- In practice, HPCToolkit fails to construct GPU CCTs for Mercury
 - Unfeasibly slow post-mortem analysis, reconstructs contexts for only 1 instruction per second!
 - Why? CCT explodes in size during analysis!
 - Need to distribute performance ~500K ways, for each instruction in one function
 - \rightarrow Need a better data representation that won't explode

Novel Graph-based Data Representation

- Directed (Calling) Context Graph
 - Generalization of other representations
 - Nodes = unique measured contexts
 - Edges = control flow (e.g. calls)
 - Edge weights = apportion between callers
 - Node values = X (Y)
 - X = Node exclusive cost
 - Y = Node inclusive cost
- Efficient for both CPU & GPU performance
 - ...But with no visible distinction between the two





Top-down Analysis Example





Bottom-up Analysis Example





Estimated Size Reduction

- Quicksilver: proxy app for Mercury
 - 36 functions (nodes in the DCCG)
 - 58 nodes in the static CCT
- Estimated size reduction: 37.9% (Assuming uniform instruction count per function)



Static call graph of one GPU kernel



(Estimate using actual instruction count per function)



Estimated Size Reduction

- Quicksilver w/o optimization
 - 145 functions (nodes in the DCCG)
 - 409 nodes in the static CCT
- Estimated size reduction: 64.5% (Assuming uniform instruction count per function)



(Estimate using actual instruction count per function)



Static call graph of one GPU kernel



Estimated Size Reduction

HPSF



- ~2K functions (nodes in the DCCG)
- ~500K nodes in the static CCT (for one function)
- Estimated size reduction: 99.6%? (Estimate using rough guess based on limited info)



(Complete guess based on left estimate)





- Novel enhancements to performance data representation
 - Same data, but in a smaller size
- Adjustments to improve parallelism in post-mortem analysis
 - Same high-level structure, but written faster
- Ongoing work

Parallel Bottleneck Of hpcprof-mpi





Removing Communication

HPSF

- Class 1: unique identifiers
 - Mostly for CCT nodes
 - All ranks use the same id to refer to a CCT node
- CCT nodes are unified based on "unique" info
 - Caller node, instruction offset, source line, etc.
 - Only merge nodes that match exactly
- We only need consensus... what about hashes?
 - Same context \rightarrow same hash in all MPI ranks
 - Hashing local data is much faster than MPI synchronization/communication
- ...But only useful if hashes don't collide



Effective Identifiers Using Hashes



- We can avoid hash conflict resolution in practice, assuming:
 - Maximum limit of ~4 billion calling contexts
 - Based on HPCToolkit's current 32-bit identifiers
 - Each node is a hash of "its (unique) location in the graph"
- 128-bit hashes (statistically) never conflict (w/ >99.9999% probability)
 - Used for communication and validation during post-mortem analysis
- 64-bit hashes are expected to have a small number of conflicts
 - Used to identify each graph node and edge on-disk
 - Disambiguate contexts at read time using graph traversal
- For performance, always leave hash conflicts unresolved
 - If a conflict causes validation errors, just restart the process (<0.0000001% chance)
 - Otherwise leave it be, disambiguate at read time

Est. Speedup of Post-mortem Analysis









- Novel enhancements to performance data representation
 - Same data, but in a smaller size
- Adjustments to improve parallelism in post-mortem analysis
 - Same high-level structure, but written faster
- Ongoing work

HPCToolkit's Performance Atlas



- Next evolution of the HPCToolkit database format
- Development is ongoing
 - Atlas library implementation: ~60%
 - Integration with hpcprof: <5%
 - Integration with hpcviewer: 0%
- Will be available as a separate library
 - First-party support for external clients



Recap

- Novel enhancements to performance data representation
 - New graph-based representation for performance data
 - Expect significantly smaller than CCT-based representation
- Adjustments to improve parallelism in post-mortem analysis
 - Hash-based identifiers to avoid MPI communication during analysis
 - Expect ~20% speedup over MPI consensus algorithm
- Atlas library implementation is ongoing
 - Making steady progress towards initial working implementation
 - Will be available as a separate library to ease adoption





000000

Backup Slides

. ...

Determining Hash Sizes



- 128-bit hashes are sufficient to identify 2^{32} items uniquely with >99.9999% probability:
 - a. Actually only need 83 bits but round up to 128 for technical convenience
 - b. Need at least 32 bits for 2³² distinct hash values
 - **C.** With 32×2 bits, probability of collision among 2^{32} items P $\approx 50\% = 2^{-1}$
 - From approximate solution to the generalized Birthday Problem
 - d. Adding 19 bits multiplies collision probability by 2⁻¹⁹, so collision probability is now 2⁻²⁰
 - Each extra bit reduces probability by ~1/2
 - e. 2⁻²⁰ < 0.0000001%, so probability of no collision is >99.9999%

ff16d9b8 671ac65c 30bb8ae6 519d6fdb

32 bits to store 2³² distinct hashes, many many hash collisions ×2 bits to reduce probability of collision to ~50% +19 bits reduce probability to 2⁻²⁰ Rounded up to 128 bits for technical convenience

Identifier Hash Construction (128-bit)



- Identifier hash for a DCCG node is a Merkle hash combining ids for dominators + flat hashes of node/edge content
 - Dominators are from measurement data, e.g. CPU stack unwind, GPU launch site + instruction offset
 - Edges may not be in DCCG, instead artificial edges just for hash construction (e.g. GPU offload)



Identifier Hash Construction (64-bit)



- Identifier hash for a DCCG node is a Merkle hash combining ids for dominators + flat hashes of node/edge content
 - Dominators are from measurement data, e.g. CPU stack unwind, GPU launch site + instruction offset
 - Edges may not be in DCCG, instead artificial edges just for hash construction (e.g. GPU offload)



Atlas: Storage Backend-agnostic Design

- Atlas is formed out of "pages"
 - Independent contiguous data blobs
 - Structured data, serialized w/ FlatBuffers (zero-copy)
 - Pages can be stored together or separate
- Pages refer to each other via descriptors
 - Can be anything: file offset, index, content hash, etc.
 - Heavily inspired by OCI images, but more general
- (Will be) widely compatible across storage media
 - POSIX/Lustre file
 - DAOS/Rabbit object store
 - OCI registry
 - NoSQL/key-value database
 - ...etc.



