A Large Ion Collider Experiment



Benchmarks for the ITS cluster finder

Sylvain Chapeland – ALICE DAQ / O²



ALICE ITS upgrade and O2 Asian Workshop 2014 | 17.06.2014

Outline

- ALICE upgrade
- ITS cluster finder algorithm
- Implementation and optimization

Tools, platforms, benchmarks

• Plans

ALICE upgrade

• O² is in design phase

250 First Level Processors (readout) : Sub-event processing, ~5x compression

1250 Event Processing Nodes: Global processing and event building (synchronous and asynchronous) Raw data discarded



 Benchmark required for estimation of computing resources needs and possible hardware

Benchmarks for ALICE upgrade

- Several ongoing activities (e.g. in CWG5)
 - raw benchmarks (cpu, memory, etc)
 - Reuse existing code (e.g. extensive experience with TPC tracking on GPUs by ALICE HLT, used in production)
- We present here one case study, with following goals:
 - Find a realistic workload
 - Implement the algorithm with various hw/sw
 - Get performance results
 - Get experience with the tools and platforms

ALICE ITS detector upgrade

- Inner Tracking System, silicon detector
 - Chip size 1650x500 pixels (inner layer)
 - 24120 chips in 7 layers
- Readout 50-400 kHz
- 40GB/s for Pb-Pb@50kHz



Example simulated event on a chip



Mostly empty

Simulated event (close-up)



Each white pixel is a hit, and a group of adjacent hits is a cluster

ITS cluster finder algorithm

- Simple to understand, no physics involved
 - Identify group of adjacent pixels
 - Compute their center of gravity
- Simple data set
 - Input: list of hits coordinates

int[], ordered by (row,column) in detectors electronics. In average 360 hits per chip, i.e. ~3kB

Output: list of cluster coordinates

float[], in millimeters from chip center). In average 60 clusters per chip, i.e. ~0.5 kB



ITS cluster finder algorithm

- Good candidate for benchmark
 - This is a good representative of one demanding type of computation to be done online, similar things done in other detectors
 - Simulated data available for input
 - Reference algorithm already available in the offline framework for output crosscheck
 - Can be easily re-implemented standalone, no external libs required
 - Fine level of parallelism for free (event or chip level)
 - Small data size should fit most architectures



C implementation for x86

- Loop over (ordered) list of hits
- Create and assign cluster ID
 - We keep current and previous pixel line in memory, with id of cluster
 - Neighborhood hits check by array indexing + bitmask (no loop)
- Group clusters
- Compute CoG
- No threading in "processEvent" code, rather 1 thread per event basis or per module
- -O3 flag with gcc 4.4.7 & icc 14.0.2

Development cycle

- Ref data: 50 events 430 modules
- Implement as simple as possible
 - started with plain C
- Verify validity of result
 - Several iterations needed
 - Some nasty use-cases
 - Added PNG debug function
- Optimize 1 thread
- Try multithread (1 thread per event/module)



Profiling

- Valgrind / callgrind
 - Good enough to find hotspots
 - Readily available
 - Heavy execution time
 - Kcachegrind GUI (kdesdk RPM) to check results
- Vtune
 - Ampl-xe gui nice, extensive threading support
 - Kernel module easy to recompile
 - Need root access to load module
 - Disable NMI: echo 0 > /proc/sys/kernel/nmi_watchdog
 - Fine details in results, many counters/metrics
 - Need most recent HW for all perf counters
- For both: easy to isolate code to be measured
- Missing tool: bookkeeping (run test, keep ref code, document findings and results)

#include <val grind/call grind. h>
CALLGRIND_START_INSTRUMENTATION
 myCode()
CALLGRIND_STOP_INSTRUMENTATION

<pre>#include <ittnotify.h></ittnotify.h></pre>	
itt_resume()	
myCode()	
itt_pause()	



Callgrind / kcachegrind

callgrind.out.41404 [/src/test1]										
<u>File View</u>	<u>G</u> o <u>S</u> ettin	gs <u>H</u> elp								
Open Image: Compared Sector Forward Image: Compared Sector Forward Image: Compared Sector Fetch										
Parts Overview B × findClusters										
findClusters Types Callers All Callers Callee Map Source Code # Ir Source ('/afs/cern.ch/ /itsCluster.test1.c') •										
Flat Profile										
<u>S</u> earch:			(No Grouping)	•	677 1	.9.31	newLineCluste	ers[k]=-1;		
Incl.	Self	Called Function	Location		670		}			
100.00	0.00	(0) main	test1: itsCluster.test1.c		6079	<i>l</i> *				_
100.00	99.68	432 findClusters	test1: itsCluster.test1.c		#	Ir	Her		Assembly Instructions	
0.13	0.13	864 📕 GI_memset	libc-2.12.so		40 1441	0.01	49.89.6	mov	%ray %r14	[
0.11	0.01	864 ■ posix_memalign	libc-2.12.so		40 1441	0.01	7e 23	ile	401469 < findClusters + 0x1d9>	
0.11	0.01	864 memalign	libc-2.12.so		40 1446	0.01	66 2e 0f 1f 84 0	0.00 nonw	%cs:0x0(%rax %rax 1)	
0.10	0.02	864int_memalign	libc-2.12.so		40 1440	0.01	00 00 00	o oo nopn	, sestox 8(/ stax, / stax, 1)	
0.09	0.05	1 296int_malloc	libc-2.12.so		40 1450	19.31	c7 03 ff ff ff ff	movl	\$0xffffffff (%rbx)	
0.05	0.05	2 263int_free	libc-2.12.so		40 1456	19.31	83 c2 01	add	\$0x1.%edx	
0.04	0.00	432 malloc	libc-2.12.so		40 1459	19.31	48 83 c3 04	add	\$0x4.%rbx	
0.04	0.04	864 malloc_consolidate	libc-2.12.so		40 1450	19.31	41 39 55 00	cmp	%edx.0x0(%r13)	
0.03	0.00	864 Ifree	libc-2.12.so		40 1461	19.31	7f ed	ia	401450 < findClusters + 0x1c0 >	
0.00	0.00	4 <u>■_</u> dl_runtime_resolve	ld-2.12.so		40 1463	0.01	49 89 6	mov	%rax.%r14	
0.00	0.00	4 <u>■</u> _dl_fixup	ld-2.12.so		40 1466	0.01	4c 89 d0	mov	%r10.%rax	
0.00	0.00	4 <u>■</u> _dl_lookup_symbol_x	ld-2.12.so		10 1460	0.04	11 05 20		(9/r0) 9/obp	التح
0.00	0.00	4 <u></u> do_lookup_x	ld-2.12.so	- 1						
4					Parts	Callees	Call Graph	All Callees	Caller Map Machine Code	
callgrind.out.	41404 [1-2	2] - Total Instruction Fetch Cost: 38	5 892 785		-					

96.5% in this loop!

Vtune – amplxe-gui

Project Navigator	× Am										=
🕼 /afs/cern.ch/user/c/chapelan/intel/amplxe/pr		Weicome routian X								c 20	-
Its	8 A	dvanced Hotspots Hotspots viewpoint (cha	nge) @					li	ntel VI une Amplifier X	E 20	113
▷ 🏧 its ref	⊲ [€	🕽 Analysis Target 🔥 Analysis Type 📰 Collection Log 🕅 Summ	ary 🔹 Bottom-up 🔹 Caller/Callee 🔹	Top-dowr	n Tree 🚦	🛃 Tasks and Fr	ames	🚯 itsCluster.te 🛛			
 [20] its_sandybridge		urse Assembly 📰 🖼 🧑 🖓 🖓 🖓	v grouping: Addross								
💯 test	50		y grouping. Address	-	0					0	
🗢 💹 test7	Sou	Source	CPU Time: Total by Utilization	≥	Â	Address	Sou	Assembly			Ê
💯 r000ah	Line		📋 Idle 📕 Poor 📙 Ok 📕 Ideal 📒 Over		🛛 Idle		Line	·,			
	789					0x40509a	795	shr \$0x1d, %r10d		0.1	
	790	int maxk;				0x40509e	795	add %r15d, %r10d			
	791	maxk=maxXnew/4+1;				0x4050a1	795	sar \$0x3, %r10d		0.2	2
	792	//maxk=szL/4;				0x4050a5	795	movsxd %r10d, %r10		1.0	2
	793					0x4050a8	795	inc %r10		0.1	
	794	#if (PACKING_MODE==1)				0x4050ab	799	<u>jle 0x4050e8 <block 27=""></block></u>			
	795	maxk=maxXnew/8+1;	2.667ms		2.66	0x4050ad		Block 25:			
	796	#endit				0x4050ad	799	pcmpeqd %xmm0, %xmm0		0.3	
	797					0x4050b1		Block 26:			-
	798	// #pragma prefetch p_fastinit:0:256	15.540	_		0x4050b1	799	add \$0x8, %r9		0.0	
	799	TOP (INT K=0;K <maxk;k+=8) td="" {<=""><td>15.542ms</td><td></td><td>15.54</td><td>0x4050b5</td><td>800</td><td>movdqax %xmm0, (%rs1)</td><td></td><td>1.7</td><td></td></maxk;k+=8)>	15.542ms		15.54	0x4050b5	800	movdqax %xmm0, (%rs1)		1.7	
	800	p_rastinit[k]=v_rastinit;	1.708ms		1.70	0x4050b9	801	movdqax %xmm0, 0x10(%rs1)		15	
	801	p_rastinit[k+1]=v_rastinit;	15.458ms		15.45	0x4050be	802	movdqax %xmm0, 0x20(%rsi)		11	-
	802	p_lastinit[k+2]=v_lastinit;	11.00/ms	_	11.00	0x4050c3	804	movdqax %xmm0, 0x30(%rsi)		13	2
	804	p_lastinit[k+3]=v_lastinit;	14.582mc		14.59	0x4050c8	805	movdqax %xmm0, 0x40(%isi)		14	
	805	p_lastinit[k+5]=v_lastinit;	14.167ms		14.50	0x4050d2	806	movddax %xmm0, 0x50(%rsi)		13	
	806	p_rastinit[k+6]=v_rastinit;	13 250ms		13 25	0x4050d2	807	movdqax %xmm0, 0x00(%rsi)		13	2
	807	<pre>p_lastinit[k+7]=v_lastinit;</pre>	13.000ms		13.00	0x4050dr	799	add \$0x80 %rsi		14	1 -
	808	}	15.000115		15.00	0x4050e3	799	cmp %r10 %r9		0.3	
	809	1				0x4050e6	799	il 0x4050b1 <block 26=""></block>		0.5	
	810	//printf("maxX=%d, maxk=%d\n".maxXnew.maxk):				0x4050e8	100	Block 27:			
	811	/* #pragma unroll(8)			-	0x4050e8	833	xor %r15d. %r15d			10
	812	for (int k=0:k <maxk:k++) td="" {<=""><td></td><td></td><td></td><td>0x4050eb</td><td>835</td><td>mov %r8d. %r14d</td><td></td><td></td><td></td></maxk:k++)>				0x4050eb	835	mov %r8d. %r14d			
	813	<pre>//p fastinit[k]= mm set epi32(-1,-1,-1,-1);</pre>				0x4050ee		Block 28:			
	814	<pre>p fastinit[k]=v fastinit;</pre>				0x4050ee	839	movl (%r13), %r9d		5.6	5
	815	}				0x4050f2	840	cmp %r15d, %r9d		5.9	
	816	*/				0x4050f5	840	cmovnl %r9d, %r15d		1.5	5
	817	/*				0x4050f9	843	test %r9d, %r9d		5.0	
	818	for (int k=0;k <maxk;k+=8) td="" {<=""><td></td><td></td><td></td><td>0x4050fc</td><td>843</td><td>jle 0x4053dc <block 49=""></block></td><td></td><td></td><td>1E</td></maxk;k+=8)>				0x4050fc	843	jle 0x4053dc <block 49=""></block>			1E
	819	<pre>mm store sil28(&p fastinit[k], v fastinit);</pre>				0x405102		Block 29:			
	820	<pre>mm_store_sil28(&p_fastinit[k+1], v_fastinit)</pre>				0x405102	845	lea -0x1(%r9), %eax		0.0	
	821	<pre>mm_store_si128(&p_fastinit[k+2], v_fastinit;</pre>	1;			0x405106	845	mov %eax, %r10d			
	822	_mm_store_si128(&p_fastinit[k+3], v_fastinit;				0x405109	845	sar \$0x1, %r10d			
	823	_mm_store_si128(&p_fastinit[k+4], v_fastinit;	1;			0x40510c	845	movsxd %r10d, %r10			
	824	_mm_store_si128(&p_fastinit[k+5], v_fastinit;);			0x40510f	845	movl (%rbx,%r10,4), %r10d			
	825	_mm_store_si128(&p_fastinit[k+6], v_fastinit	(;			0x405113	845	mov %r10d, %r11d			
	826	_mm_store_si128(&p_fastinit[k+7], v_fastinit;	(g			0x405116	845	sar \$0x10, %r11d			
	827	}				0x40511a	845	test \$0x1, %al			
	000	selected 1 row(s	5):	1.708ms	1.70 🗸	040511.	0.45		Highlighted 1 row(s):	: 1.7	
) <						>		Ľ
		No filters are applied 🛛 🖉 Apy Brososs	aroad Apy Modulo		A.m.	Utilization		Only user function in Joline Meder on	Functions only		IS I

Vtune – amplxe-gui

<u>Ma</u>		adurant draaming	/project	s/test7 - Inte	l VTune Amplifier (on pcal	dbuild0	5.cern.c	h)							_ • ×
roject Navigator	× 🗛 🔥	🖄 🕅 💿 🕨 🖨 🐠 Welcome	r001ge X												=
/afs/cern.ch/user/c/chapelan/intel/amplxe/pr.			valeration viewagint (shanga) @								lote		molifier V	C 2012
Maits	Gen	eral exploration General E	xploration viewpoint (<u>change</u>) ©								IIII	er v rune A		2013
its ref	An	alysis Target 🔥 Analysis Type 🔛 Collec	tion Log 🛚 🛍 Summary 🛛 🖧 Bot	tom-up 💁 Toj	p-down Tree 🛛 🗮 Tasks and Frai	mes 🛛 🗋 i	tsCluster	:te 🛛							
🦉 its sandybridge				Address											
Am test	Source	Assembly	Assembly grouping:	Address											
✓ // test7			Hardware Event Count: Tota	Hardware	Hardware Event Count: Total by	/ Har			Filled Pipeli	ne Slots:	Filled Pipeli	ine Slots:	Unfilled Pip	eline Slot	Unfi^
r000ah	Sou	Course		*			CPI	CPI	\gg	>	2	>	2	≫	
👰 r001ge	Line	Source	CPU_CLK_UNHALTED. THREAD	CPU_CL THREAD	INST_RETIRED.ANY	INS ANY	Total	Self	Retiring	Bad Specul	Retiring	Bad Specul	Back-e Bound	Front-e Bound	Back Bot
	956	clusters[ix].rawY=clusters[ix].Y	0.5%	5,200,000	600,000	600	8.667	8.667				L.			
	736	Y=hits[i].Y; // keep track of c	0.5%	6,000,000	16,400,000	16,	0.366	0.366					_		_
	766	prevLineY=-1;	0.5%	6,200,000	13,200,000	13,	0.470	0.470						<u> </u>	
	771	prevLineClusters=newLineClus	0.5%	6,200,000	12,800,000	12,	0.484	0.484		_		_		<u> </u>	
	701	clusters[i].joinCluster=-1;	0.6%	6,400,000	7,400,000	7,4	0.865	0.865					_		
	775	<pre>tmpmaxX=maxXnew;</pre>	0.6%	6,600,000	9,800,000	9,8	0.673	0.673							
	884	if (prevLineY!=-1) {	0.6%	6,600,000	11,200,000	11,	0.589	0.589					1		
	849	clusters[cid].X+=X;	0.6%	6,800,000	14,200,000	14,	0.479	0.479							
	696	clusters[i].X=0;	0.6%	7,000,000	8,000,000	8,0	0.875	0.875	<u> </u>					_	
	795	<pre>maxk=maxXnew/8+1;</pre>	0.6%	7,000,000	10,400,000	10,	0.673	0.673							
	848	clusters[cid].nPix++; // add	0.6%	7,000,000	19,200,000	19,	0.365	0.365							
	695	for (int i=0;i <maxclusters;i++) td="" {<=""><td>0.7%</td><td>7,600,000</td><td>6,400,000</td><td>6,4</td><td>1.188</td><td>1.188</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></maxclusters;i++)>	0.7%	7,600,000	6,400,000	6,4	1.188	1.188							
	697	clusters[i].Y=0;	0.7%	7,600,000	10,000,000	10,	0.760	0.760							
	699	clusters[i].rawX=-1;	0.7%	8,200,000	15,400,000	15,	0.532	0.532							
	700	clusters[i].rawY=-1;	0.7%	8,400,000	7,000,000	7,0	1.200	1.200							
	765	if (Y!=newLineY+1) { // curren	0.9%	10,000,000	15,800,000	15,	0.633	0.633							_
	846	if (cid!=-1) { // existing clu	0.9%	10,200,000	16,400,000	16,	0.622	0.622							
	843	if (X>0) {	0.9%	10,600,000	15,600,000	15,	0.679	0.679						<u> </u>	
	698	clusters[i].nPix=0;	1.1%	12,000,000	19,600,000	19,	0.612	0.612		_		_		<u> </u>	
	738	if (Y!=newLineY) { // we have c	1.1%	12,000,000	6,400,000	6,4	1.875	1.875							
	839	X=hits[i].X; // keep track of c	1.1%	12,400,000	17,200,000	17,	0.721	0.721							
	858	if ((newcid!=-1)&&(newcid!	1.3%	14,800,000	24,200,000	24,	0.612	0.612				U			
	955	clusters[ix].rawX=clusters[ix].X	1.4%	15,600,000	1,000,000	1,0	15.600	15.600							
	840	if (X>maxXnew) {maxXnew=X;}	1.4%	15,800,000	17,400,000	17,	0.908	0.908							
N	734	for (int i=0;i <nhits;i++) td="" {<=""><td>1.4%</td><td>16,200,000</td><td>26,200,000</td><td>26,</td><td>0.618</td><td>0.618</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></nhits;i++)>	1.4%	16,200,000	26,200,000	26,	0.618	0.618							
N,	898	if (debug) printf ("creating clu	1.4%	16,200,000	12,400,000	12,	1.306	1.306							
	850	clusters[cid].Y+=Y;	1.4%	16,400,000	25,400,000	25,	0.646	0.646							
	960	clusters[ix].Y=(dead_bottom + (c	1.8%	20,600,000	29,800,000	29,	0.691	0.691		_		_			
	803	<pre>p_fastinit[k+3]=v_fastinit;</pre>	2.2%	25,200,000	35,200,000	35,	0.716	0.716							
	804	p_fastinit[k+4]=v_fastinit;	2.2%	25,200,000	33,400,000	33,	0.754	0.754				<u> </u>		<u> </u>	
	799	for (int k=0;k <maxk;k+=8) td="" {<=""><td>2.5%</td><td>28,400,000</td><td>112,400,000</td><td>112</td><td>0.253</td><td>0.253</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></maxk;k+=8)>	2.5%	28,400,000	112,400,000	112	0.253	0.253							
	806	p_fastinit[k+6]=v_fastinit;	2.5%	29,000,000	31,200,000	31,	0.929	0.929							
	807	<pre>p_fastinit[k+7]=v_fastinit;</pre>	2.6%	30,000,000	35,600,000	35,	0.843	0.843		I.		I.		<u> </u>	
	802	<pre>p_fastinit[k+2]=v_fastinit;</pre>	2.7%	30,400,000	29,400,000	29,	1.034	1.034	<u> </u>	l		U			
	805	<pre>p_fastinit[k+5]=v_fastinit;</pre>	2.7%	30,600,000	31,200,000	31,	0.981	0.981						l	
	801	<pre>p_fastinit[k+1]=v_fastinit;</pre>	2.8%	32,000,000	43,200,000	43,	0.741	0.741		I .		I		<u> </u>	
	959	clusters[ix].X=(dead_left + (clu	2.9%	33,400,000	27,600,000	27,	1.210	1.210			U	U		I	
		Selected 1 row(s):	0.0%												
		>							_		_				
	🗙 No fi	ilters are applied. 🛛 🐺 🛛 Any Process	 Any Thread 	\sim	Any Module 🗸 🗸	Only	iser func	tions 🗸	Inline Mod	le: on	✓ Function	ns only	$\mathbf{\sim}$		

Code optimization (1 thread)

• Hotspot analysis shows that most time is spent in clearing index line

(id of cluster for each pixel in current line)

for (int k=0; k<szX; k++) {
 newLineClusters[k]=-1;</pre>

- Optimization
 - 1. Dummy loop
 - 2. + Don't reset full line (keep max index updated)
 - 3. Use memset /* works only because (int)-1 = char[4] {-1,-1,-1} */
 - 4. Use 128bits intrinsics => movdqa
 - 5. + Manual loop unroll (8)
 - 6. + Pack data manually (1 coord per int)
 - 7. + Pack data manually (2 coords per int)
 - 8. + Pack data manually (3 coords per int)

```
#include <immintrin.h>
__m128i v_fastinit;
v_fastinit=_mm_set_epi32(-1, -1, -1, -1);
for (int k=0; k<maxX/4+1; k++) {
  ((__m128i *)(newLineClusters))[k]=v_fastinit;
}</pre>
```

Performance, events per second 1 thread, E5 2665 2.4GHz



Code optimization (1 thread)

- Observed so far:
 - Pack data = improve performance, because algorithm is memory I/O bound
 - Importance of data structs (e.g. aligned to use vector instructions)
 - Assembly useful (e.g. to check the good AVX instructions are used in the end)
 - Some obvious and simple things might still be worth optimizing "by hand"
- Among ideas tried:
 - Automatic loop unroll with icc pragma
 - Pack 3 coords in one int but loose time on division
 - Use short instead of int does not help (data not packed automatically)
 - Walk back array faster than resetting indices
 - Excellent performance and ease of implementation with c++11 vector class
 - Use one array for cluster index + 1 bitmask for upper row neighbor check is the best solution so far
 - Optimization of CoG computation helpful (another 10-15%)
- Demanding process...
 - Algorithm was looking like a piece of cake, but reality is different
 - Requires effort and iterations
 - Quite addictive

Scalability (04/2014)

Events processed per second

Westmere i5-680@3.6GHz, 2 cores, HT

Events processed per second

SandyBridge E5 2665 @2.4GHz, 16 cores, HT



1 thread per event, 50 events

Scalability (observations 04/2014)

- Observations
 - Suspect bottleneck in memory access, high back-end usage
 - Was not expecting good results because of lightweight computing task (rather memory bound)
 - Effects on turbo mode affects the measurements with low thread count (and the linear baseline for scalability check)
 - Fluctuations with high thread count because of small data set and variance in processing time per event
- Algorithm scales decently
 - Suspect locks in memory allocation from 1-thread hotspot analysis
 - Overhead of threading / workload distribution
 - Need to deal with streaming data in NUMA node (data/CPU affinity)



1 thread per event (or chip?)

What changed (04->06 / 2014)

- Bitmask for neighbor check
 - i.e. 24*64 bit in addition to 1500 x 32 bit index
- C++ 11
 - Std::vector
 - Threading
- Still to do:
 - NUMA affinity handling (standalone experiments with libnuma are promising)
 - Redo profiling on ivy bridge (compared to sandybridge, gives finer details on back-end, higher level metrics and hints for memory bottlenecks)
 - Plot processing time per chip versus number of hits on chip (probably linear)

Latest performance (yesterday)



Proposal on data format

- Now: hit1 (X,Y) hit2 (X,Y) ... [order by row,col]
- What about grouping consecutive pixels in a row?
- Proposed: hitgroup1 (X1,X2,Y) hitgroup (X1,X2,Y)
 ... [still ordered by row, col]
 - Should be easy implement in FEE
- On reference data set, this represents a 2.5x compression, which in turns also improves throughput in algorithm

${\scriptstyle \mathsf{inaccurate \& provocative}} \ \ COSt \ \ \mathsf{observations}$

CPU	CPU Price \$	CPU mark	Perf event/s	Price/perf \$/(event/sec)
I5-680 2c@3.6 GHz (desktop Q2 2010)	320	3539	373	0.86
E5-2665 16c@2.4 GHz (server Q2 2012)	1420	12452	724	1.96

- Higher clock speed is still one easy way to get perf
 to be considered for specific low-latency needs ?
- For some workloads, a desktop might still be a good contender and more than twice cheaper at same perf level

X86 summary

- We can now process 430 inner chips @ 400Hz for 1 thread / 5KHz for 1 machine (6 GB/s with 32bit coords)
- How does this extrapolate to full detector? (need full data set) 10-50 times more?
- We can still progress on local scalability
 - NUMA
 - Threading framework
 - Data format
 - Wider vectors
- 10KHz/machine (for 400 chips) looks reasonably reachable, probably more depending on data format
- Still far from 50KHz for 20000 chips... would need 50-250 nodes just for cluster finding?
- Algorithm mature enough to clarify estimate, but depends much on realistic data format. We should focus on this now.

ITS Cluster Finding on GPU

- Work from Boonyarit Changaival @ KMUTT
- Exploit multicore architecture
- Use one thread to process one hit (in most kernel)



GPU

- Preliminary results
 - i5-3470 @ 3.2GHz
 - Nvidia Geforce GTX780

- **30** events/second (same ref. data as for x86, i.e. 430 modules)
 - 1 CPU thread
 - Show full occupancy over all kernels (>90%)
 - Very low serialization portion (in most kernels)

GPU Runtime Overview

- Spend 71% of total runtime on GPU
- Overhead is 29% of the total runtime
 - Initialization
 - Host and device memory allocation
 - Moving data between host and device
- Why slower than x86?
 - Copying memory between Host and Device
 - Loops in GPU decrease performance
 - Not many rich instructions
- My personal analysis: this algorithm does not fit well GPU, too little floating point math & vector, too much I/O
- Still some ideas to try: multiple GPU cards, multiple CPU threads
- We will still work a bit on this in coming weeks, but unlikely that we gain 2 orders of magnitude...

Xeon Phi

- Initial tests promising for code portability
 - Ok to compile, doc not so easy to start with e.g. TBB
 - Simple programming paradigm to adapt the code: C++11 / TBB / parallel_for #pragma offload_attribute (push, target(mic)) #include "tbb/tbb.h" #pragma offload_attribute(pop)

```
__attribute__((target(mic))) void ComputeClusters(dataIn dIn, dataOut *dOut) {...}
```

```
void ParallelApplyComputeClusters( dataIn* in, dataOut *out, size_t n ) {
    parallel_for( blocked_range<size_t>(0, n),
        [=](const blocked_range<size_t>& r) {
        for(size_t i=r. begin(); i!=r. end(); ++i)
            ComputeClusters(in[i], &out[i]);
        }
    );
}
```

- Work paused, need to clarify scalability first
- Then tune code with hw specific features (e.g. wider vectors)
- However it might be that we need next generation, with out of order execution able to handle the many branching we have in cluster finding code. I/O bandwidth also to be tested.
- HW made available by CERN Techlab 17/06/2014 S.Chapeland / Platform Benchmarking for ALICE upgrade

What's next

- Get full detector data set
- Continue platform implementation

– x86

- Vector/bitmask
- Haswell (new 256bit int instructions, BSR & gather/scatter)
- NUMA affinity / memory pre-allocate / data streaming – i.e. "framework", not algorithm
- MIC 512bit instructions, 70 "slow" cores
- GPU any good idea?
- FPGA (many implementations, c.f. RT14)?

Conclusion

One need a real use case to progress!

- Find a realistic workload
- Implement the algorithm with various hw/sw
- Get performance results
- Get experience with the tools and platforms
- Exercise was fruitful in all aspects