High-Performance MPI Application Energy Consumption Profiling

Victor Getmanskiy

Oleg Shapovalov

Efim Sergeev

Dmitry Kryzhanovsky

Singularis Lab, Ltd.

Volgograd State Technical University

Outline

1. Power management in Processors
2. Energy consumption profiling
3. Packages and benchmarks
4. Difficulties and reefs
5. How to profile: methodology
6. Analysis results
7. Conclusions
8. Common results

Energy-efficient supercomputers

20-th place Dell C8220X Cluster, Intel Xeon E5-2680v2 10C 2.8GHz, InfiniBand 4x FDR, Intel Xeon Phi 7120P

2.39 GFlops/W

Intel® Xeon Phi™

1-st place ASUS ESC4000 FDR/G2S, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR, AMD FirePro S9150

5.27 GFlops/W

Scope of power analysis

- Hardware analysis
  - analysis of computational system consumption
  - no analysis of single application consumption
- Software analysis at application level (Power Top)
  - analysis of single application consumption
  - detecting the consumption sources (IO, memory, HDD, CPU etc.)
- Software analysis at function level (Intel® VTune Amplifier)
  - analysis of consumption of single functions and units
  - detecting the functions (methods) requiring energy consumption optimization by the developer

C-states of Intel® Xeon Phi™

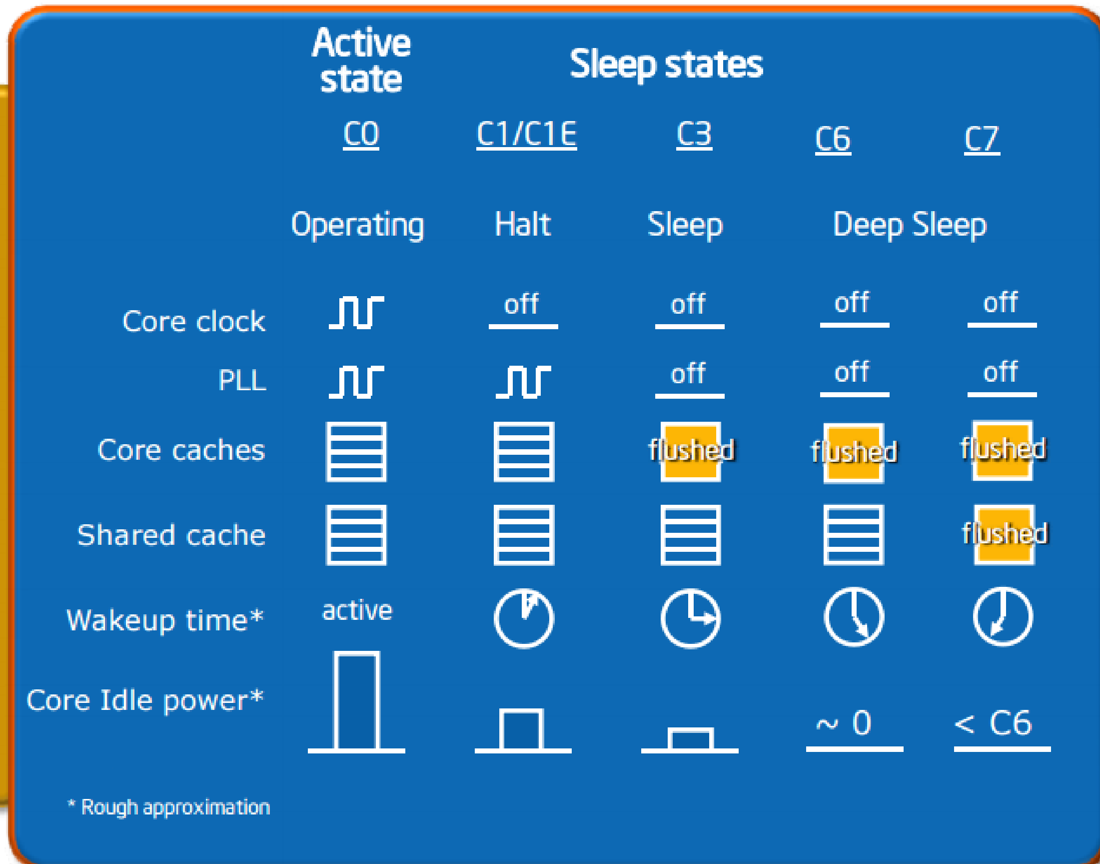
CPU **C-states** are core power states requested by the Operating System Directed Power Management (OSPM) infrastructure. C1-Cn states describe states where the processor clock is inactive and different parts of the processor are powered down.

C0	Full on
C1/C1E	Auto-halt
C3 Auto/Deep	Sleep
C6	Deep Power Down

C-states of Intel® Xeon Phi™

Processor C States

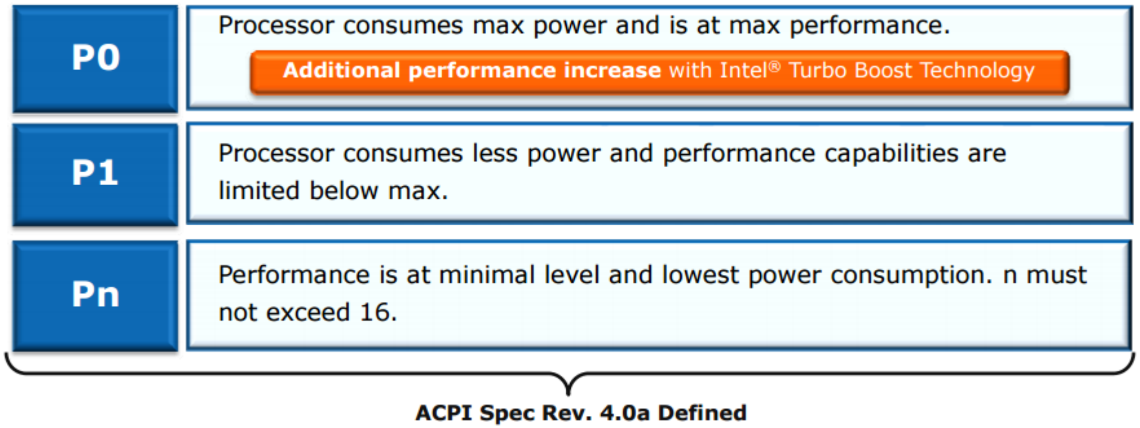
- C-States ensure lower processor power during idle light workloads
- C-State limits can be set by BIOS
- A processor can go into sleep states several thousand times per second
- OS controls the C states in its idle process



P-states and C-states

C-states – Processor Power States
P-states – Processor Performance Power States
P-states apply only to C0 state of processor and control voltage and clock frequency

Processor Performance Power States (P-States)



Multiple voltage and frequency operating points

- Software controlled by writing to MSRs
- The voltage is optimized based on the selected frequency and number of active processor cores
- All active processor cores share the same frequency and voltage.

Number of supported states is processor dependent

Power states summary

	P-state	C-state
Processor	+ (P)	+ (PC)
Core	-	+ (C)
Logical	-	+ (LC)

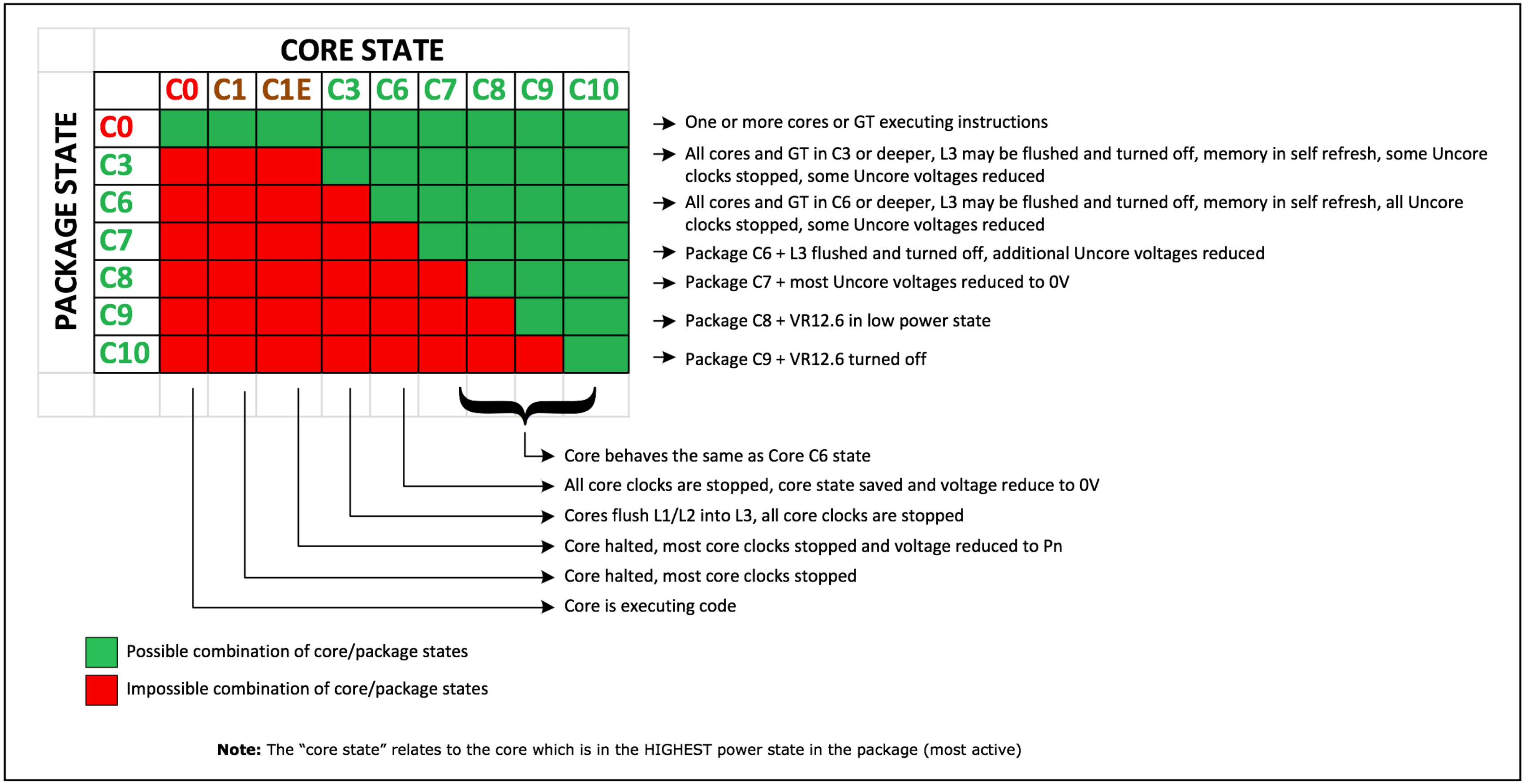
Processor C-state = Min(core C-states)

Core C-state = Minimum barrier(set of all logical C-states)

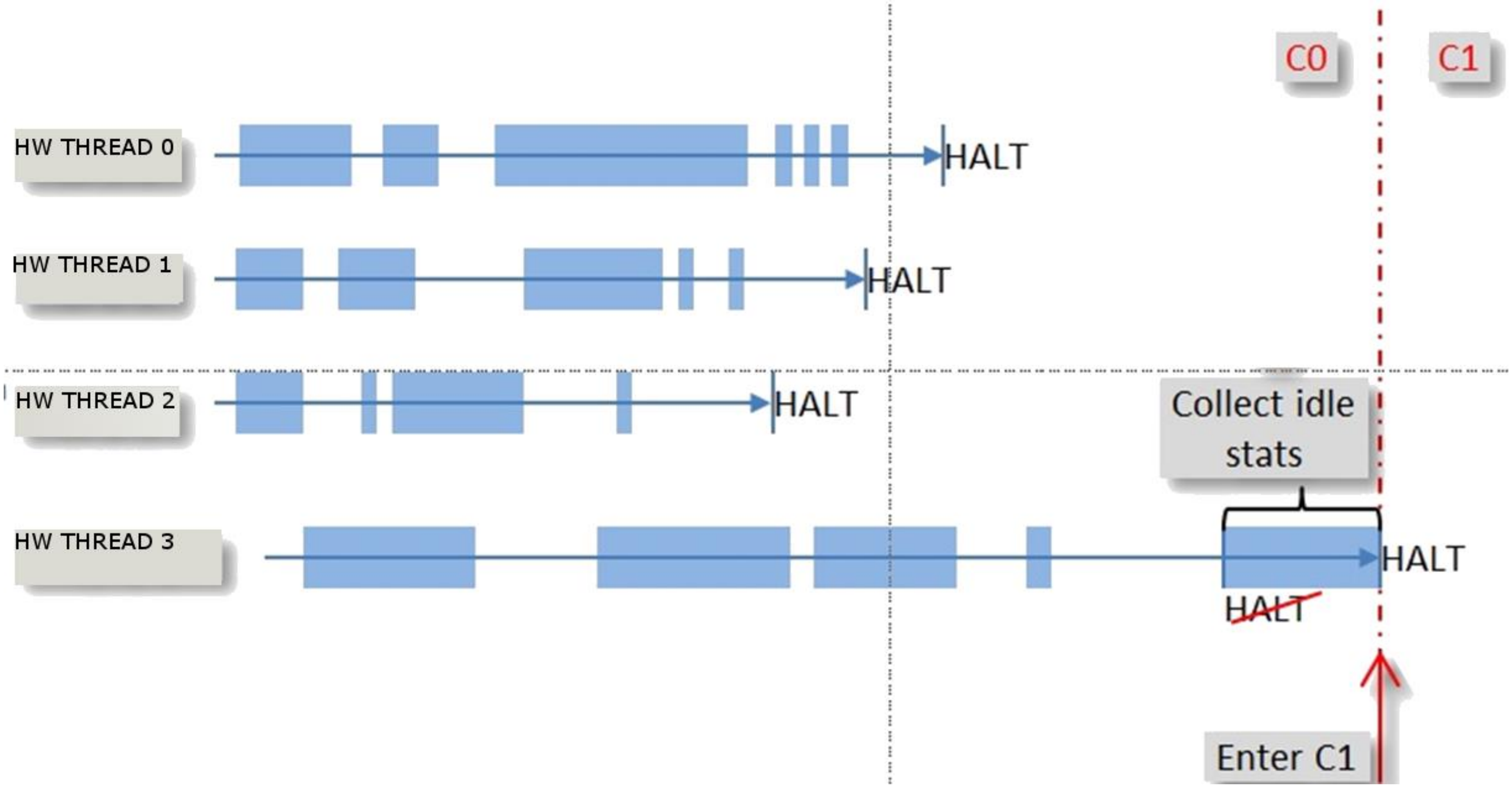
Logical C-state = anything the OS wants (hardware threads "states" on Intel Xeon Phi)

C-states (Haswell Mobile Processor)

Processor Package and Core C-States

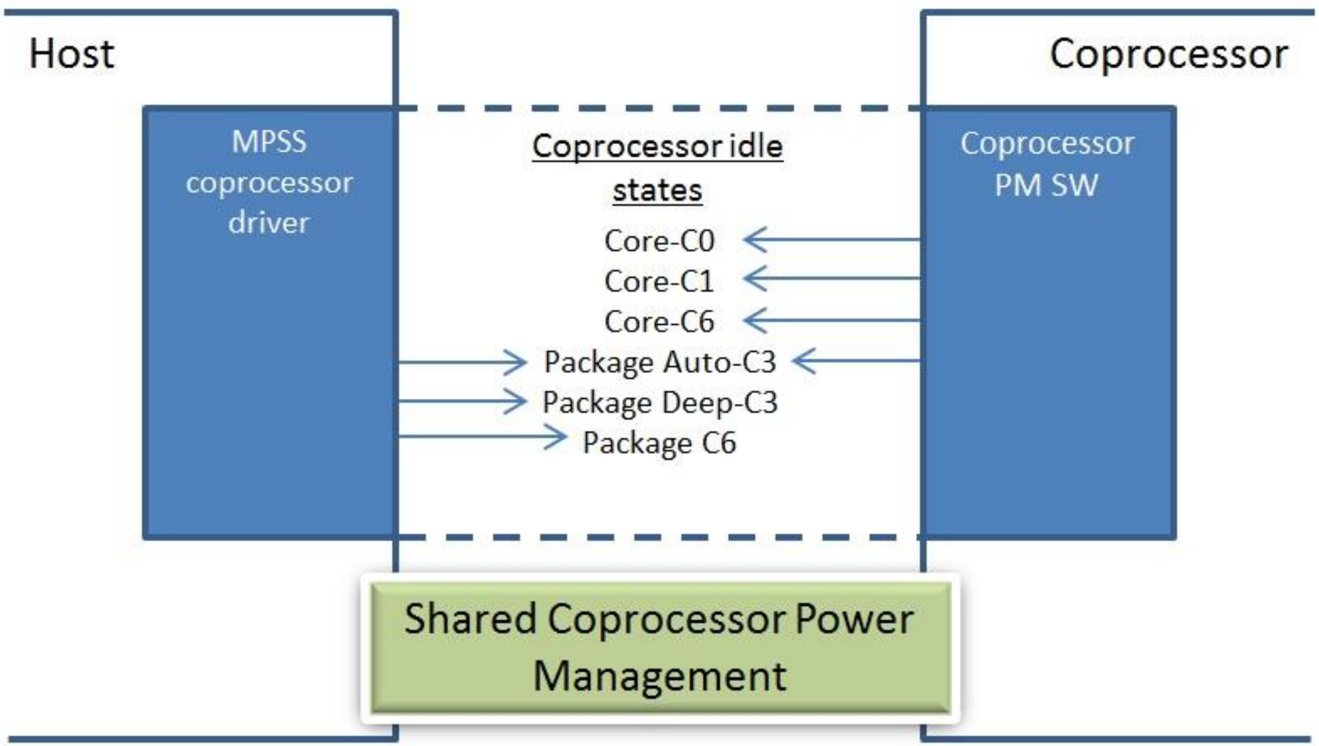


Switching to Package IDLE state



C-states (Intel® Xeon Phi™)

	C0	C1	C3	C6
PC0				
PC3 AUTO				
PC3 DEEP				
PC6				



Intel® Xeon Phi™ C-states In-Depth

Package Idle State	Core State	Uncore State	TSC/LAPIC	C3WakeupTimer	PCI Express* Traffic
Auto C3 (initiated by PM SW can be overriden by MPSS)	Preserved	Preserved	Frozen	On expiration, package exits PC3	Package exits PC3
Deep C3	Preserved	Preserved	Frozen	No effect	Times out
PC6	Lost	Lost	Reset	No effect	Times out

Package Auto-C3: Ring and Uncore clock gated
Package Deep-C3: VccP reduced
Package C6: VccP is off (I.e. Cores, Ring and Uncore are powered down)

Software tools

	OS	CPU models	CPU consumption	Consumption of the other devices	Linkage with the code
Power Top	Linux, Solaris	Core2Duo …	Modelled	Modelled	No
Joulemeter	Windows	Core2Duo …	Modelled	Modelled	No
Intel Power Gadget	Windows, Linux, OS X	Core-i, XEON (Sandy Bridge) …	Direct measurement	–	No
XCode 5 Power Profiler	OS X	Core-i, XEON (Sandy Bridge) …	Combined indices	–	Statistics for single threads
Intel ® VTune Amplifier	Windows, Linux	Core-i, XEON (Sandy Bridge) …	Direct measurement	–	Yes

Intel® VTune Amplifier is the most comfortable tool for developers

Packages and benchmarks

Open source and popular in industry HPC applications:

- **GAMESS** – computational quantum chemistry.
- **GROMACS** – molecular dynamics for modelling physicochemical processes.
- **LAMMPS** – simulation of the classical molecular dynamics of particle system.
- **NAMD** – an object oriented code for modelling molecular dynamics of big biomolecular systems.
- **OpenFOAM** – computational hydrodynamics and work with fields (scalar, vector, tensor).

Difficulties and reefs

1. Building the packages with various implementations of MPI.
2. Need to balance between the size of task (sample), the size of application profile and its level of detail.
3. Problems with detecting units under dynamic linking and detecting functions (methods) inside units.

Difficulties and reefs

5. Collecting PMU events is limited with only one profiling process.
6. Detecting the mode of profiling for one and all processes and the necessity in analysis of MPI scheduler energy consumption.
7. Adjusting hardware for maximum scalability of the applications (NUMA, hyper-threading).

Difficulties: dynamic MPI linking

Advanced Hotspots Hardware Event Counts viewpoint (change)

Analysis Target | Analysis Type | Summary | PMU Events | Caller/Callee | Top-down Tree | Tasks and Frames

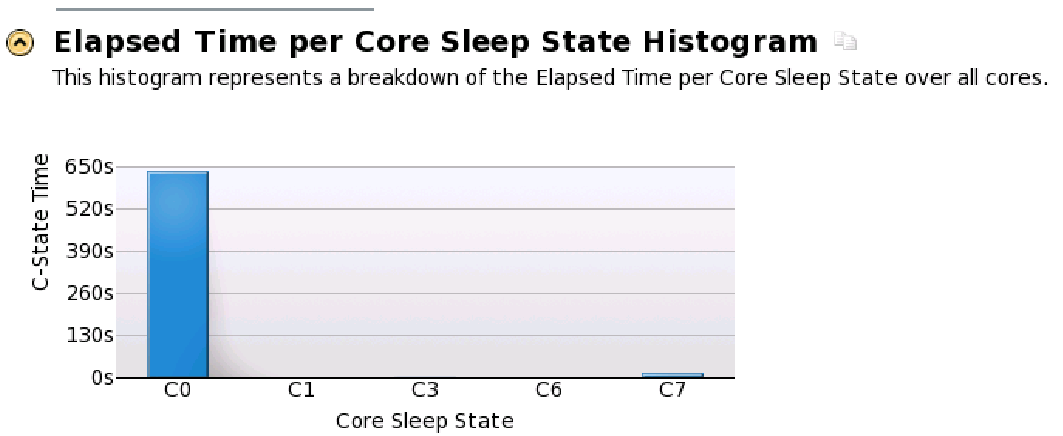
Grouping: Module / Function / Call Stack

Module / Function / Call Stack	Hardware Event Count by Hardware Event Type		
	Energy Core	Energy Pack	Energy DRAM
namd2	13,262,674,064	16,642,949,488	469,946,384
[Unknown]	417,066,448	522,906,672	14,126,768
vmlinux	300,799,840	386,657,728	11,820,304
libmpi.so.4.1	270,844,064	339,839,936	9,358,560
func@0xd0150	175,018,896	219,609,904	6,047,600
MPIDI_CH3I_Progress	80,318,144	100,763,760	2,787,328
__I_MPI__intel_ssse3_rep_memcpy	4,101,904	5,150,928	133,920
MPIDI_CH3_iSendv	2,053,728	2,580,160	69,952
MPID_nem_tcp_connpoll	1,631,216	2,047,744	52,448
MPI_Iprobe	1,584,016	1,985,088	53,632
MPID_nem_tcp_poll	1,119,296	1,402,960	38,832
MPID_nem_network_poll	630,288	793,632	24,672
MPIDI_nem_active_vc	580,944	730,560	22,272
PMPI_Wtime	523,632	657,680	16,064
MPID_nem_tcp_vc_active	473,712	592,784	17,920
MPID_Iprobe	467,712	586,832	15,904
func@0x2b5df0	286,464	360,208	8,512
MPIDI_CH3U_Recvq_FU	232,224	290,960	6,400
MPID_Isend	222,320	282,032	8,000
MPID_nem_tcp_iStartContigMsg	175,136	219,952	6,080
PMPI_Recv	120,752	150,528	5,888
MPID_Wtime_todouble	119,456	149,536	4,096
MPIR_Test_impl	118,016	147,856	3,968
MPI_Isend	117,744	147,056	6,272
MPID_Recv	111,248	141,584	3,776
func@0x205ba0	63,024	78,784	1,600
MPID_nem_tcp_iSendContig	62,960	78,720	1,536
PMPI_Get_count	62,272	77,184	2,112

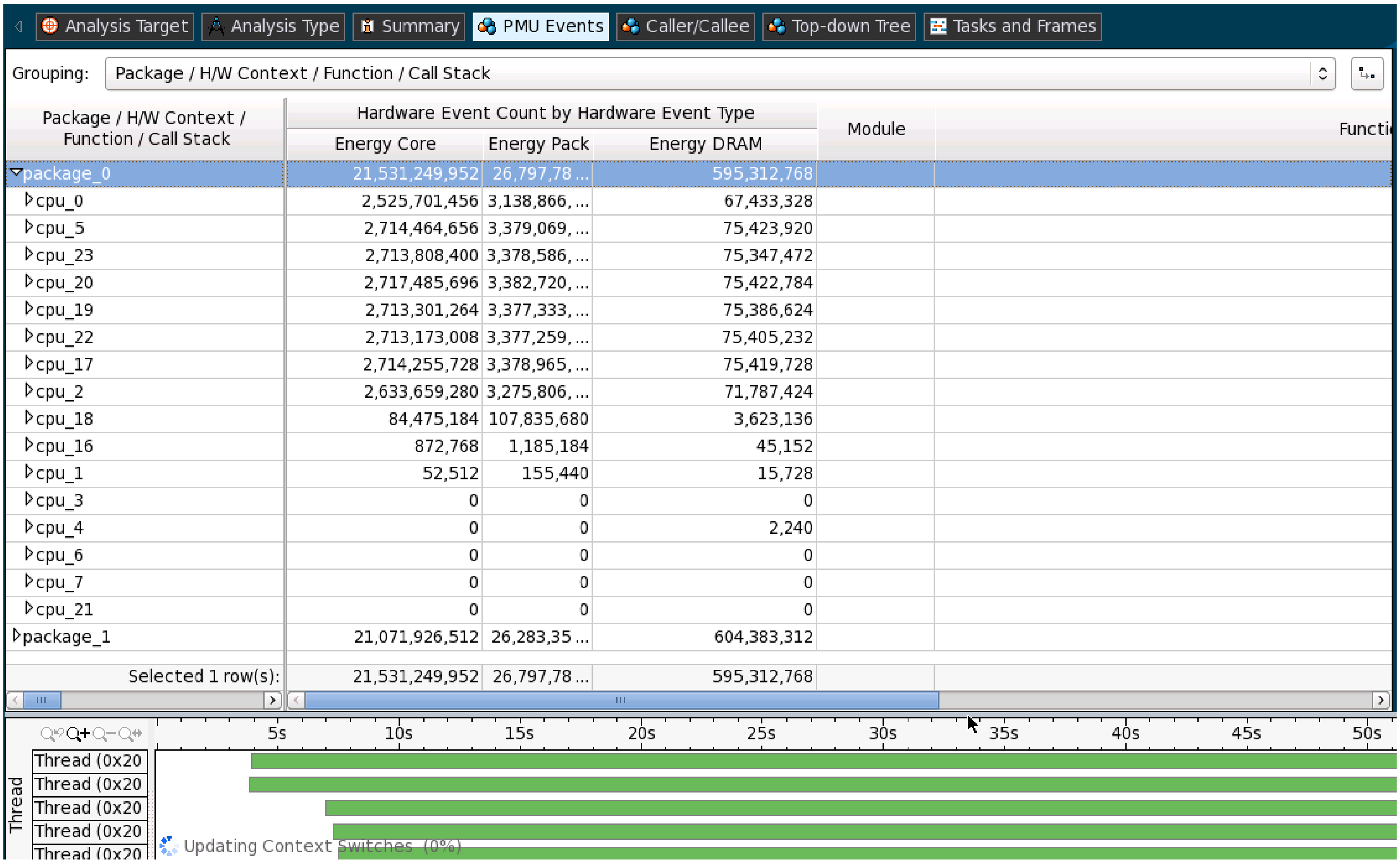
Difficulties: profiling mode for all / one process

NAMD

LAMMPS

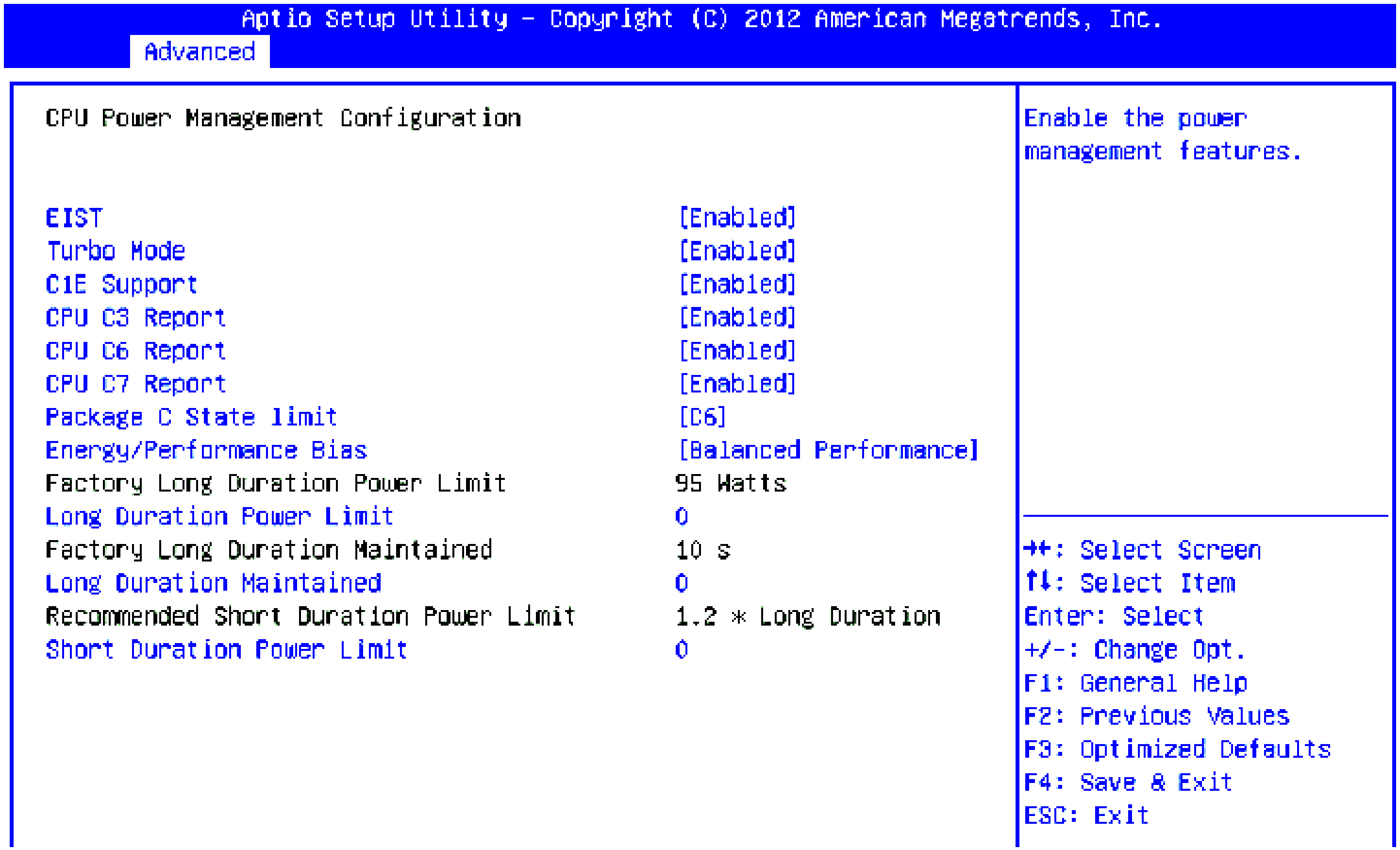


Difficulties: profiling mode for all / one process



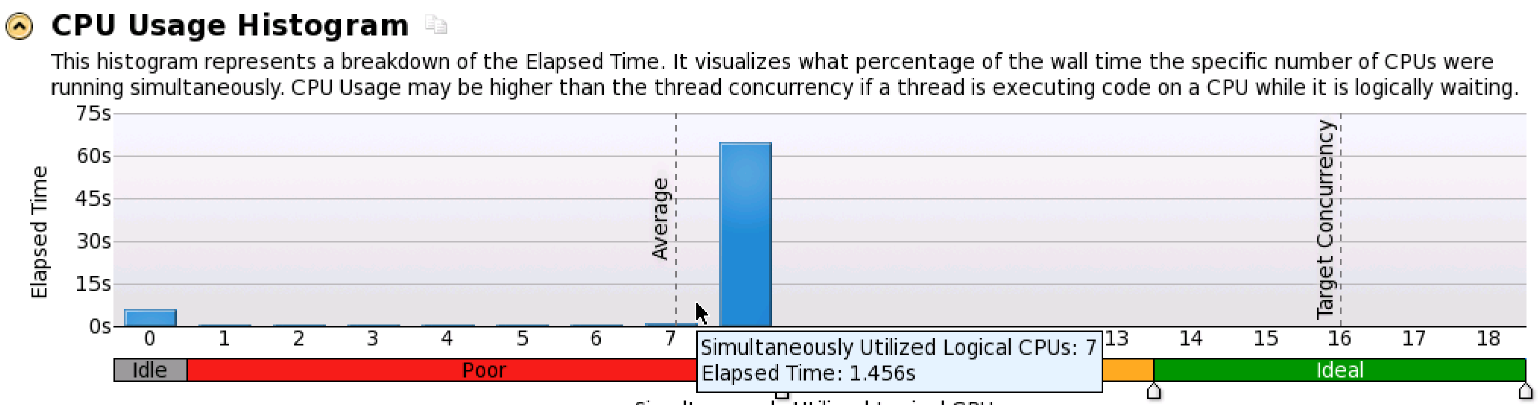
Package / H/W Context / Function / Call Stack	Hardware Event Count by Hardware Event Type: Energy Core	Energy Pack	Energy DRAM	Module	Functi
package_0	21,531,249,952	26,797,78 ...	595,312,768		
cpu_0	2,525,701,456	3,138,866, ...	67,433,328		
cpu_5	2,714,464,656	3,379,069, ...	75,423,920		
cpu_23	2,713,808,400	3,378,586, ...	75,347,472		
cpu_20	2,717,485,696	3,382,720, ...	75,422,784		
cpu_19	2,713,301,264	3,377,333, ...	75,386,624		
cpu_22	2,713,173,008	3,377,259, ...	75,405,232		
cpu_17	2,714,255,728	3,378,965, ...	75,419,728		
cpu_2	2,633,659,280	3,275,806, ...	71,787,424		
cpu_18	84,475,184	107,835,680	3,623,136		
cpu_16	872,768	1,185,184	45,152		
cpu_1	52,512	155,440	15,728		
cpu_3	0	0	0		
cpu_4	0	0	2,240		
cpu_6	0	0	0		
cpu_7	0	0	0		
cpu_21	0	0	0		
package_1	21,071,926,512	26,283,35 ...	604,383,312		
Selected 1 row(s):	21,531,249,952	26,797,78 ...	595,312,768		

Difficulties: adjusting hardware (NUMA, hyberthreading)

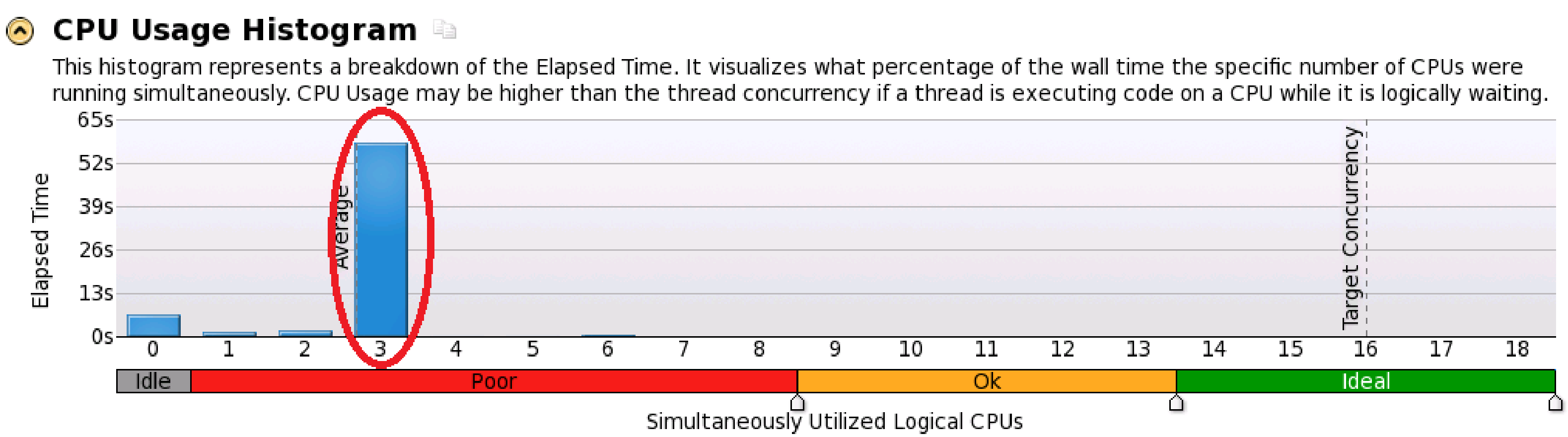


Difficulties: adjusting hardware (NUMA, hyberthreading)

Without NUMA



With NUMA



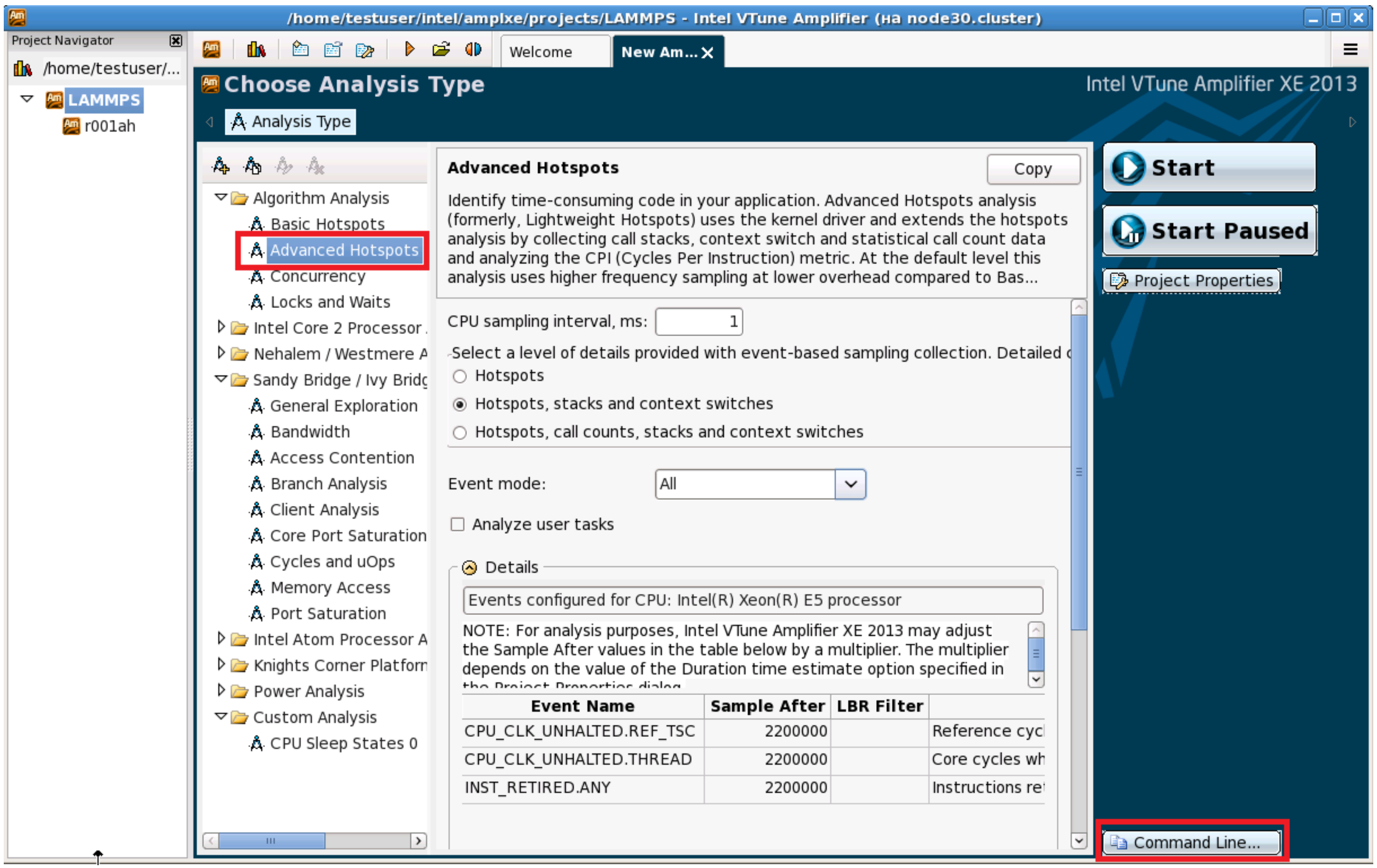
How to profile

Methodology:

1. Use static linking.
2. Profile all CPUs at node.
3. Find a relative part of the scheduler energy consumption.
4. Profile one of the processes being run.
5. Detect the MPI library functions in the application unit.
6. Estimate the energy consumption relative parts of the application functions and MPI library.

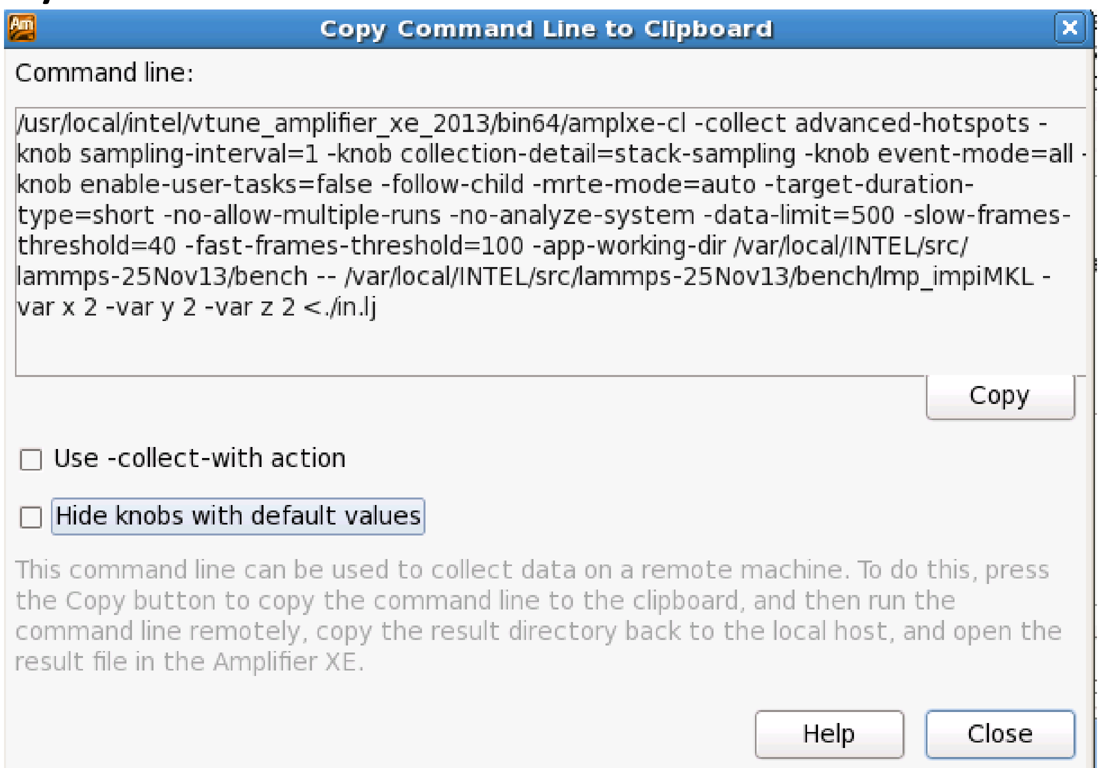
How to run analyses

- Profiling must be run in the mode <u>Advanced Hotspot Analysis</u>.
- To run a profiler GUI use `amplxe-gui`.



How to see the command line

- If you click the button Command Line…, you will see the command to run a console profiler.
- If you reset the check-box Hide knobs default values , the default keys will be added to the command line.



Adjusting the environment

.bashrc

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
          . /etc/bashrc
fi
# User specific aliases and functions
export PATH=/usr/gcc/gcc470/bin:$PATH
export LD_LIBRARY_PATH=/usr/gcc/gcc470/lib64:/usr/gcc/gcc470/lib:$LD_LIBRARY_PATH
source /opt/intel/compilers/composerxe/mkl/bin/mklvars.sh intel64
source /opt/intel/compilers/composerxe/bin/compilervars.sh intel64
```

.bashrc.namd_IntelMPI

```
module unload MPI-3/MVAPICH2
module unload MPI-3/OpenMPI
module unload MPI-3/MPICH
module load MPI-3/IntelMPI
export PATH=/var/local/INTEL/install/intelmpi-icc/third-
party/src/NAMD_2.9_Source/Linux-x86_64-ics-2013-SP1:$PATH
export PATH=/var/local/INTEL/src/NAMD_2.9_Source/charm-6.4.0/bin:$PATH
```

How to run the profiling

```
#!/bin/bash
DIR=$(pwd)
WORKING_DIR=$DIR/NAMD/small

# remove profile output folder
$DIR/RemoveDir.sh $PROFILE_DIR

# rewrite bashrc
(cat $DIR/.bashrc) > ~/.bashrc
(echo source $DIR/.bashrc.namd_IntelMPI) >> ~/.bashrc
source ~/.bashrc

# run
cd $WORKING_DIR
mpirun -genv I_MPI_FABRICS=shm:tcp -genv I_MPI_WAIT_MODE=1 \
-host node31 -n 1 -wdir=$WORKING_DIR amplxe-cl -collect advanced-hotspots -knob
collection-detail=stack-sampling -r $PROFILE_DIR – namd2 apoa1 : \
-host node31 -n 15 -wdir=$WORKING_DIR namd2 apoa1 : \
-host node30 -n 16 -wdir=$WORKING_DIR namd2 apoa1
```

Some notes about running

- The folder for collecting output data specified with the key –r is added with the rank number (for example, .0). So, if we specify

  `-r ~/profiles/LAMMPS`

  we'll have the folder

  `~/profile/LAMMPS.0`
- The profiling output results should be watched via GUI.
- The profiling output data folder must be cleaned before each launching VTune Amplifier.

Power consumption metrics

- Energy Core, µJ – energy dissipated on the core
- Energy Pack, µJ – energy dissipated on the processor
- Energy DRAM, µJ – energy dissipated on the memory unit
- You need to group the functions to detect which of them consume more energy (it is to be the package being run itself)

Advanced Hotspots Hardware Event Counts viewpoint (change)

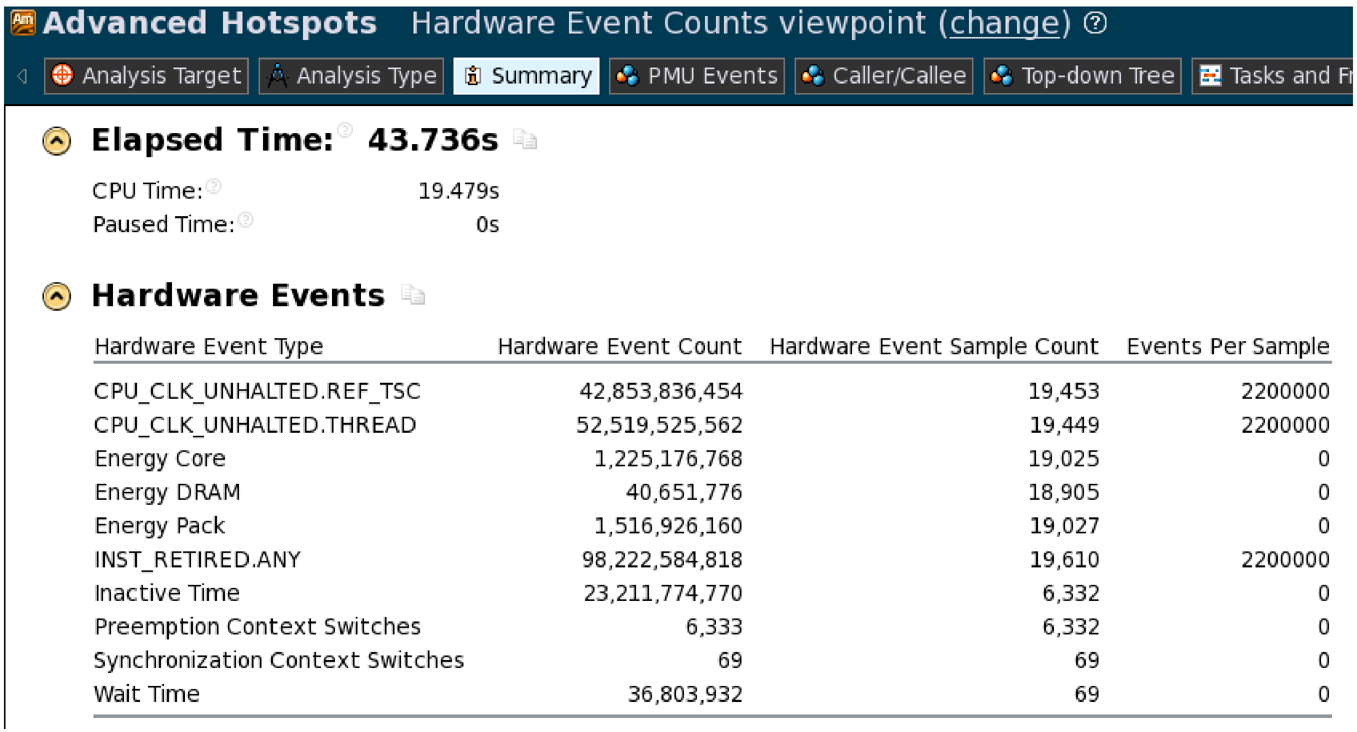
Analysis Target | Analysis Type | Summary | PMU Events | Caller/Callee | Top-down Tree | Tasks and Frames

Grouping: Module / Function / Call Stack

Module / Function / Call Stack	Hardware Event Count by Hardware Event Type			Module
	Energy Core	Energy Pack	Energy DRAM	
lmp_impiMKL	12,980,515,552	16,394,977,472	373,273,584	
LAMMPS_NS::PairLJCut::compute	7,092,074,544	9,005,436,656	202,162,704	lmp_impiMKL
LAMMPS_NS::Neighbor::half_bin_newton	2,226,706,512	2,801,350,288	70,635,696	lmp_impiMKL
LAMMPS_NS::Comm::reverse_comm	1,328,104,464	1,660,942,272	36,841,392	lmp_impiMKL
LAMMPS_NS::Comm::forward_comm	990,692,832	1,231,359,616	33,247,184	lmp_impiMKL
LAMMPS_NS::Comm::borders	445,019,248	555,523,424	13,378,176	lmp_impiMKL
LAMMPS_NS::Verlet::run	191,455,376	239,864,272	5,285,120	lmp_impiMKL
LAMMPS_NS::FixNVE::initial_integrate	127,063,952	161,298,624	1,158,240	lmp_impiMKL
LAMMPS_NS::Verlet::force_clear	88,995,872	115,051,120	611,088	lmp_impiMKL
LAMMPS_NS::FixNVE::final_integrate	83,676,672	106,114,064	1,012,416	lmp_impiMKL
LAMMPS_NS::Neighbor::check_distance	62,005,232	78,737,056	834,176	lmp_impiMKL
LAMMPS_NS::Run::command	60,336,176	75,433,648	1,531,296	lmp_impiMKL
LAMMPS_NS::Pair::sbmask	55,547,456	70,537,424	1,601,600	lmp_impiMKL
LAMMPS_NS::AtomVecAtomic::unpack_reverse	55,506,720	70,200,784	1,472,528	lmp_impiMKL
LAMMPS_NS::Neighbor::build	28,384,688	37,128,016	284,992	lmp_impiMKL

Analysis of power consumption

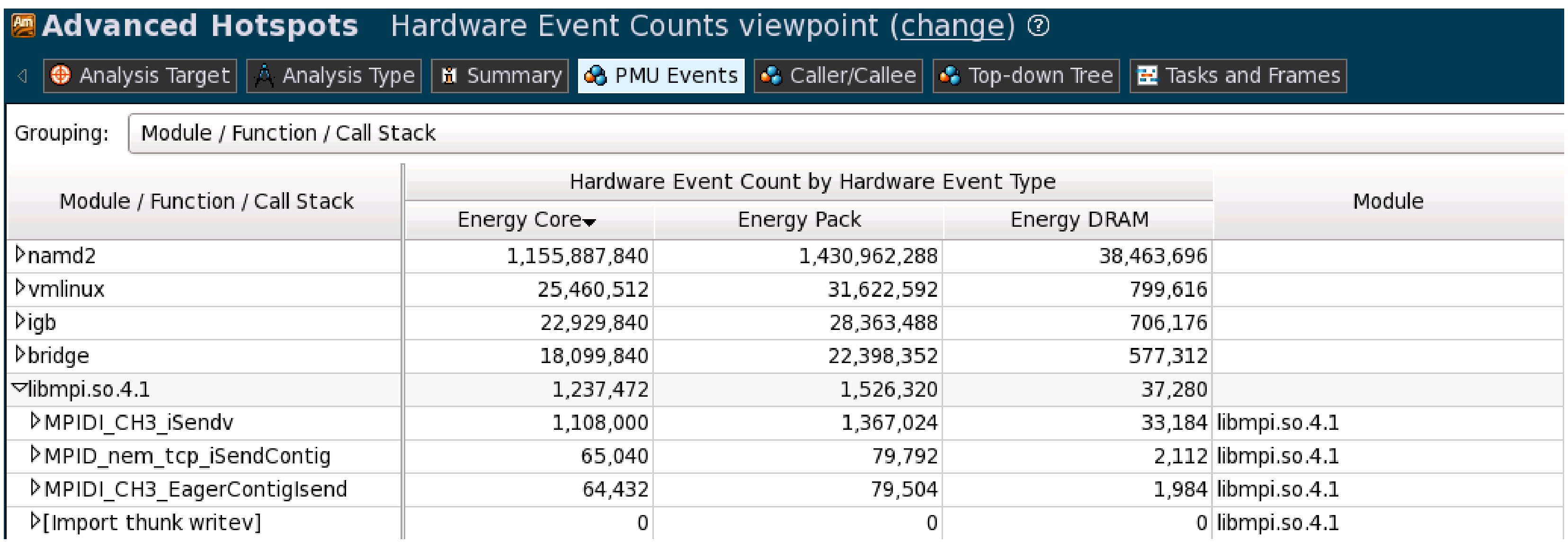
- You need to switch on the mode Hardware Events Counts Viewpoint
- The tab Summary displays the general power consumption for the period of time while we were profiling the application



Hardware Event Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample
CPU_CLK_UNHALTED.REF_TSC	42,853,836,454	19,453	2200000
CPU_CLK_UNHALTED.THREAD	52,519,525,562	19,449	2200000
Energy Core	1,225,176,768	19,025	0
Energy DRAM	40,651,776	18,905	0
Energy Pack	1,516,926,160	19,027	0
INST_RETIRED.ANY	98,222,584,818	19,610	2200000
Inactive Time	23,211,774,770	6,332	0
Preemption Context Switches	6,333	6,332	0
Synchronization Context Switches	69	69	0
Wait Time	36,803,932	69	0

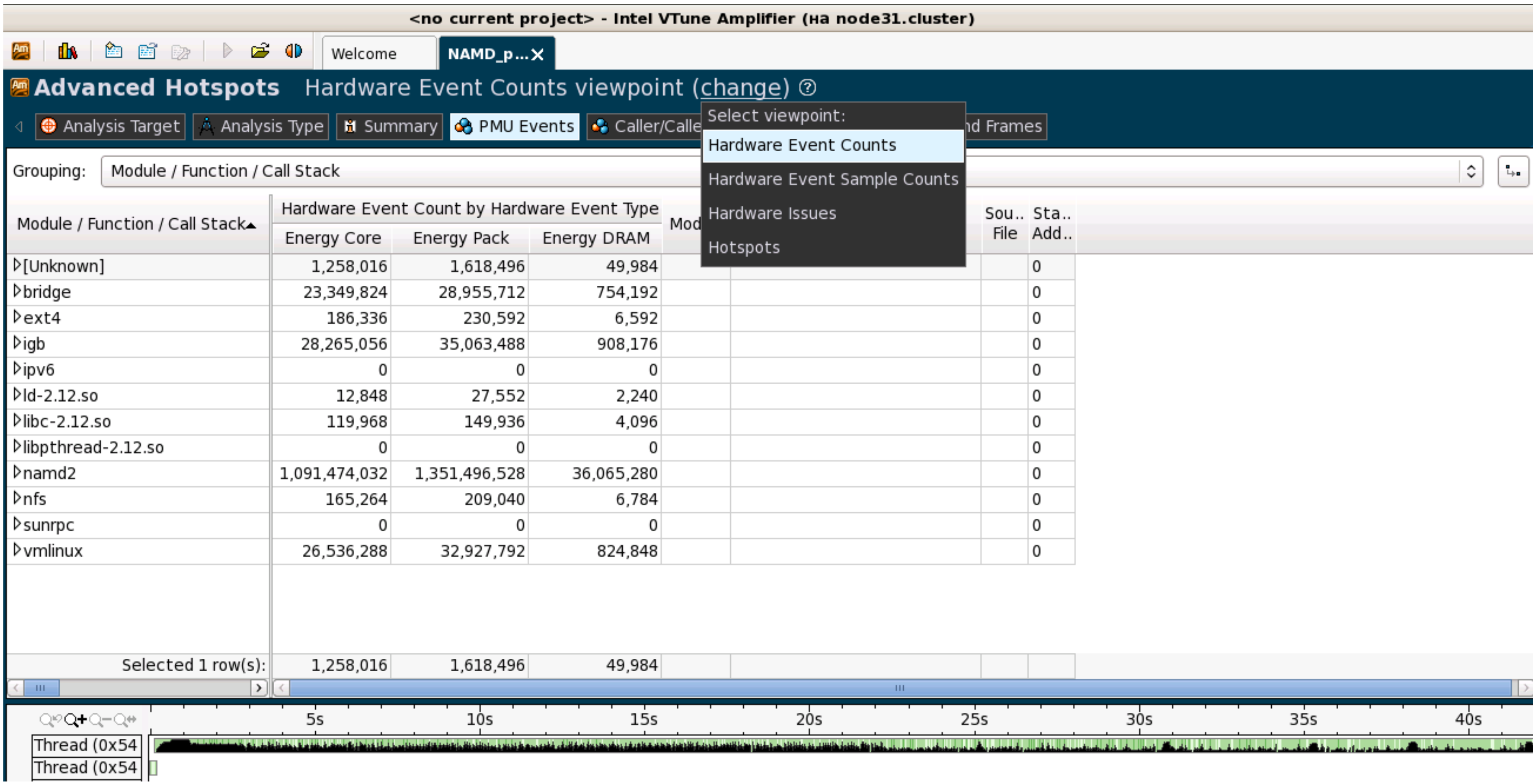
Analysis of power consumption

- To see individual consumption for groups of functions and units you can use the tab PMU Events
- You can use for grouping three parameters (Grouping:Module, Grouping:Function, Grouping:CallStack)
- If you expand the units, the most consuming functions will be displayed at the top



Module / Function / Call Stack	Hardware Event Count by Hardware Event Type			Module
	Energy Core	Energy Pack	Energy DRAM	
namd2	1,155,887,840	1,430,962,288	38,463,696	
vmlinux	25,460,512	31,622,592	799,616	
igb	22,929,840	28,363,488	706,176	
bridge	18,099,840	22,398,352	577,312	
libmpi.so.4.1	1,237,472	1,526,320	37,280	
MPIDI_CH3_iSendv	1,108,000	1,367,024	33,184	libmpi.so.4.1
MPID_nem_tcp_iSendContig	65,040	79,792	2,112	libmpi.so.4.1
MPIDI_CH3_EagerContigIsend	64,432	79,504	1,984	libmpi.so.4.1
[Import thunk writev]	0	0	0	libmpi.so.4.1

Watching the collected data



Sorting by energy consumption

Advanced Hotspots Hardware Event Counts viewpoint (change)

Analysis Target | Analysis Type | Summary | PMU Events | Caller/Callee | Top-down Tree

Grouping: Module / Function / Call Stack

Module / Function / Call Stack	Hardware Event Count by Hardware Event Type		
	Energy Core	Energy Pack	Energy DRAM
namd2	1,091,474,032	1,351,496,528	36,065,280
ComputeNonbondedUtil::calc_pair_energy	218,274,416	269,682,176	7,351,360
ComputeNonbondedUtil::calc_pair_energy_fullelect	120,518,640	149,407,728	3,799,456
MPID_nem_mpich2_test_recv	114,659,056	142,278,064	3,866,752
pairlist_from_pairlist	100,898,512	124,588,464	3,280,816
ComputeNonbondedUtil::calc_self_energy	97,986,240	120,904,912	3,174,464
ScriptTcl::run	62,542,080	77,539,312	2,030,240
MPIDI_CH3I_Progress	54,285,056	67,308,768	1,862,480
ComputeNonbondedUtil::calc_self_energy_fullelect	49,547,872	61,178,608	1,617,136
CsdScheduler	28,025,920	34,720,320	907,536
Parameters::assign_dihedral_index	27,026,336	33,420,160	996,416
MPID_Iprobe	25,093,296	31,166,688	868,112

General energy consumption

Advanced Hotspots Hardware Event Counts viewpoint (change)

Analysis Target | Analysis Type | Summary | PMU Events | Caller/Callee | Top-down Tree | Tasks and F

Elapsed Time: 42.609s

CPU Time:	18.581s
Paused Time:	0s

Hardware Events

Hardware Event Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample
CPU_CLK_UNHALTED.REF_TSC	40,877,324,510	18,555	2200000
CPU_CLK_UNHALTED.THREAD	50,122,607,951	18,574	2200000
Energy Core	1,171,367,632	18,218	0
Energy DRAM	38,622,192	18,203	0
Energy Pack	1,450,679,136	18,220	0
INST_RETIRED.ANY	90,046,098,213	18,989	2200000
Inactive Time	18,768,191,932	6,769	0
Preemption Context Switches	6,770	6,769	0
Synchronization Context Switches	1,489	1,489	0
Wait Time	2,190,496,730	1,489	0

Analysis results

Intel MPI functions with the highest energy consumption

Module / Function / Call Stack	Energy Core	Energy Pack	Energy DRAM	Energy Core	Energy Pack	Energy DRAM
MPID_nem_mpich2_test_recv	114659056	142278064	3866752	53%	53%	54%
MPIDI_CH3I_Progress	54285056	67308768	1862480	25%	25%	26%
MPID_Iprobe	25093296	31166688	868112	12%	12%	12%
__I_MPI___intel_ssse3_rep_memcpy	4829472	5981408	147552	2%	2%	2%
MPIDI_CH3_iSendv	3424624	4246848	107248	2%	2%	2%
MPI_Iprobe	3172560	3928784	1068	1%	1%	0%
MPID_nem_tcp_connpoll	2458384	3044944	70	1%	1%	0%
MPID_nem_network_poll	2080144	2577088	70368	1%	1%	1%
MPI_Recv	1605376	1997280	54144	1%	1%	1%

Energy consumption relative parts for the application functions and MPI library

Application	Energy Core	Energy Pack	Energy DRAM
namd2	1091474032	1351496528	36065280
Without MPI	873696888	1081513344	28925454
	Energy Core	Energy Pack	Energy DRAM
Total Power	1171367632	1450679136	38622192
Relative Power Consumption	Energy Core	Energy Pack	Energy DRAM
mpiexec	0%	0%	0%
MPI	19%	19%	18%
namd2	75%	75%	75%

Conclusions

LAMMPS

- Intel MPI – the lowest consumption (absolute and relative values)
- MVAPICH – the highest consumption

NAMD

- Worse scalability than for LAMMPS
- Parallel computing takes a more intensive data exchange
- <u>Possible conclusion:</u> **if the package is a good scalable, then Intel MPI gives a better (lower) consumption; if the packages is a bad scalable, Intel MPI gives worse results.**

Conclusions

GROMACS

- Lower MPI energy consumption than for LAMMPS and NAMD
- The function SENDRECV, taking the most energy consumption in GROMACS, requires less overheads and prevents CPU from switching C-state to C-7 (LAMPS and NAMD use SEND and RECV instead of SENDRECV)

GAMESS

- The relative part of MPI energy consumption is the lowest (< 1%)
- It is caused by the high part of computational code in compare with parallel processes synchronization

Conclusions

OpenFOAM

- Has the highest (and extremely high) MPI power consumption.
- Different MPI libraries show values around 50 %.
- **In the scope of energy consumption optimization OpenFOAM requires a special attention.**

Conclusions

1. Static MPI linking with one process profiling provides more correct results.
2. Running the package on several cluster nodes provides more correct and representative results.
3. Running the package on several cluster nodes under one process profiling is possible for Intel MPI, MPICH, MVAPICH and Open MPI with some differences in the MPI scheduler commands.

Conclusions

5. Analyzing energy consumption on cluster, you need to take into account that CPU is in the state C0 the most time. Switching between the states happens only if intensive data exchange takes place (as it is in NAMD).
6. Energy consumption of the application functions and MPI library depends on the used MPI implementation and the task you are solving (its sizes).

Common results

- The methodology how to profile applications for power consumption has been described and master-classes on this topic have been created.
- The benchmark parameters for more representative power profiling have been adjusted.
- The distribution of energy consumption for different MPI implementations and for single MPI functions has been estimated for several HPC packages.

Thank you for your attention

www.singularis-lab.com