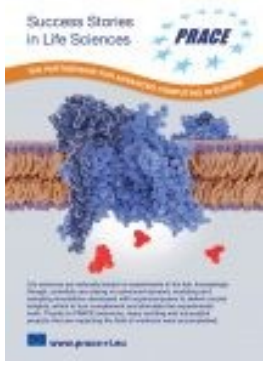
PRACE-6IP

WP7.4B BPG development activity



Ole W. Saastad, Dr. Scient.

University of Oslo / Sigma2

Overview – what is PRACE ?

- Partnership for Advanced Computing in Europe
  - The mission of PRACE is to enable high-impact scientific discovery and engineering research and development across all disciplines to enhance European competitiveness for the benefit of society. PRACE seeks to realise this mission by offering world class computing and data management resources and services through a peer review process.

What can PRACE offer ?

- Research Infrastructure
- PRACE HPC Access
- Training and Education
- HPC Market Surveillance

What can PRACE offer ?

- Reaseach infrastructure
  - PRACE is established as an international not-for-profit association (aisbl) with its seat in Brussels. It has 26 member countries whose representative organisations create a pan-European supercomputing infrastructure, providing access to computing and data management resources and services for large-scale scientific and engineering applications at the highest performance level.

What can PRACE offer ?

- PRACE HPC Access
  - PRACE systems are available to scientists and researchers from academia and industry from around the world.
    - Preparatory Access
    - SME HPC Adoption Programme in Europe (SHAPE)
    - Distributed European Computing Initiative (DECI)
    - Project Access - researchers and research groups

What can PRACE offer ?

- Training and Education
  - PRACE has an extensive education and training effort for effective use of the RI through seasonal schools, workshops and scientific and industrial seminars throughout Europe.
  - Seasonal Schools target broad HPC audiences, whereas workshops are focused on particular technologies, tools or disciplines or research areas.

What can PRACE offer ?

- HPC Market Surveillance
  - PRACE undertakes software and hardware technology initiatives with the goal of preparing for changes in technologies used in the Research Infrastructure and provide the proper tools, education and training for the user communities to adapt to those changes.

Work Package 7 – Applications Enabling and Support

- Applications Enabling Services for Preparatory Access
- DECI Management and Applications Porting
- Benchmarks Unified European Applications Benchmark Suite (UEABS)
- Best Practice Guides
- Enhancing the High Level Support Teams

Task 7.4 Supporting European HPC Researchers

- One/two BPG should be published within a year

Previous PRACE WP7 BPGs

1. **Deep Learning**, *February 2019* — ***PRACE 5IP***
2. **Modern Interconnects**, *February 2019* — ***PRACE 5IP***
3. **ARM64**, *February 2019* — ***PRACE 5IP***
4. **Parallel I/O**, *February 2019* — ***PRACE 5IP***
5. **AMD EPYC**, *February 2019* — ***PRACE 5IP***
6. **HPC for Data Science on the Cray Urika**, *January 2019* — ***PRACE 5IP***
7. **Intel Xeon Phi**, *January 2017* — ***PRACE 4IP (Update 3IP)***
8. **Haswell/Broadwell**, *January 2017* — ***PRACE 4IP***
9. **GPGPU**, *January 2017* — ***PRACE 4IP (Update 3IP)***
10. **Knights Landing**, *January 2017* — ***PRACE 4IP***

Best Practice Guides

2020 PRACE Best Practice Guides

- Application porting and code-optimization activities for European HPC systems (April 2020)
- Modern Processors (October 2020)

2019 PRACE Best Practice Guides

- AMD EPYC (February 2019)
- ARM64 (February 2019)
- Deep Learning (February 2019)
- Modern Interconnects (February 2019)
- Parallel I/O (February 2019)
- HPC for Data Science (January 2019)

2017 PRACE Best Practice Guides

- GPGPU (January 2017)
- Haswell/Broadwell (January 2017)
- Intel Xeon Phi (January 2017)
- Knights Landing (January 2017)

2014 PRACE Best Practice Guides

- Blue Gene/Q (January 2014)

2013 PRACE Best Practice Guides

- Cray XE-XC (December 2013)
- Curie (November 2013)
- IBM Power 775 (November 2013)
- Anselm (June 2013)
- Generic x86 (May 2013)
- SuperMUC (May 2013)
- Chimera (April 2013)
- Hydra (March 2013)
- JUROPA (March 2013)

Relevant BPGs – x86-64 & ARM

- Haswell/Broadwell, January 2017
- Knights Landing, January 2017
- ARM64, February 2019
- AMD EPYC, February 2019
- Modern Processors (Skylake, Rome, ARM), November 2020

- Some overlap, but different topics covered differently in each guide.

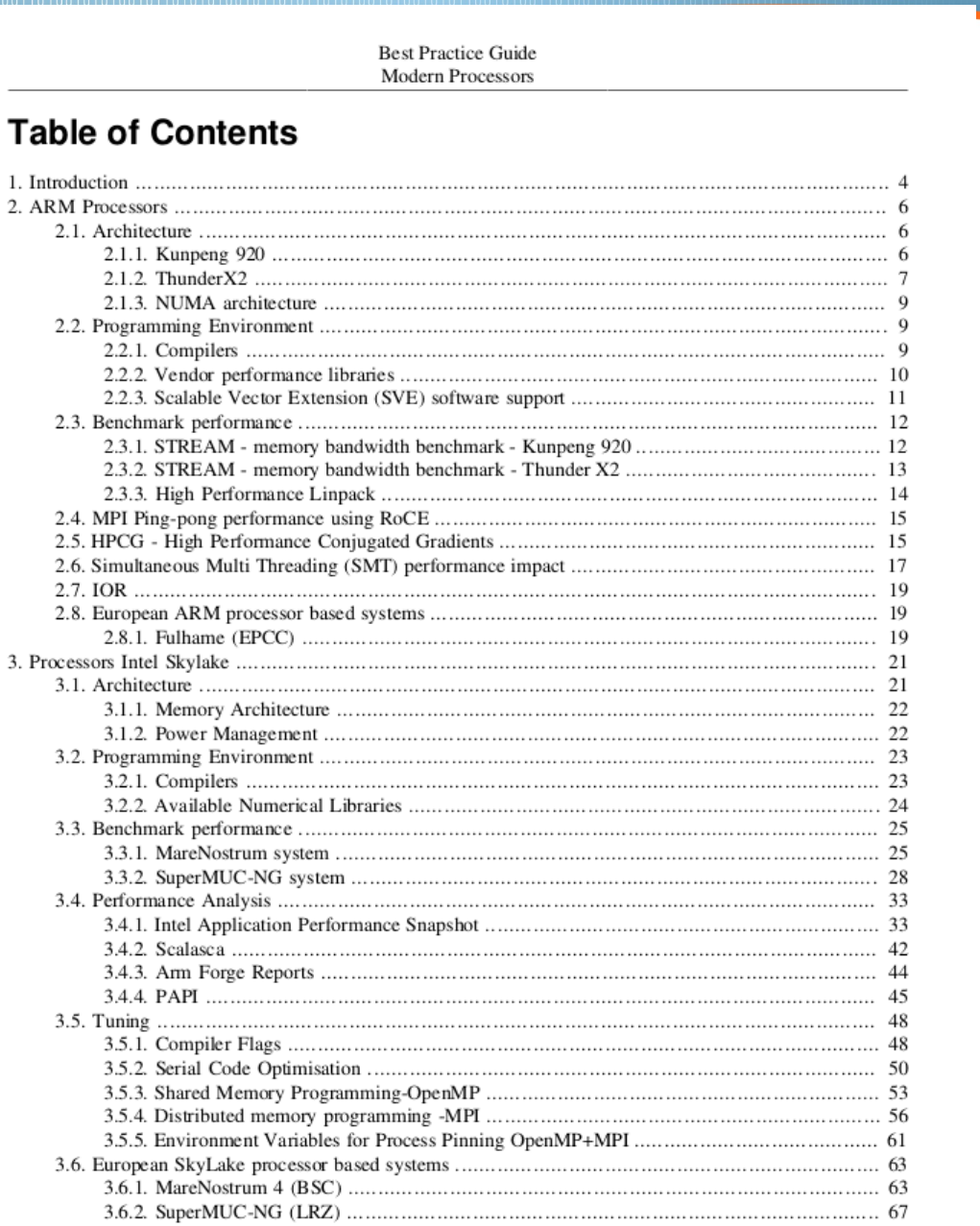
Best Practice Guide Modern Processors

Ole Widar Saastad, University of Oslo, Norway
Kristina Kapanova, NCSA, Bulgaria
Stoyan Markov, NCSA, Bulgaria
Cristian Morales, BSC, Spain
Anastasiia Shamakina, HLRS, Germany
Nick Johnson, EPCC, United Kingdom
Ezhilmathi Krishnasamy, University of Luxembourg, Luxembourg
Sebastien Varrette, University of Luxembourg, Luxembourg
Hayk Shoukourian (Editor), LRZ, Germany

23-10-2020

BPG –
a mixture of a:

- reference guide
- field guide



Best Practice Guide
Modern Processors

Table of Contents

1. Introduction 4
2. ARM Processors 6
   2.1. Architecture 6
      2.1.1. Kunpeng 920 6
      2.1.2. ThunderX2 7
      2.1.3. NUMA architecture 9
   2.2. Programming Environment 9
      2.2.1. Compilers 9
      2.2.2. Vendor performance libraries 10
      2.2.3. Scalable Vector Extension (SVE) software support 11
   2.3. Benchmark performance 12
      2.3.1. STREAM - memory bandwidth benchmark - Kunpeng 920 12
      2.3.2. STREAM - memory bandwidth benchmark - Thunder X2 13
      2.3.3. High Performance Linpack 14
   2.4. MPI Ping-pong performance using RoCE 15
   2.5. HPCG - High Performance Conjugated Gradients 15
   2.6. Simultaneous Multi Threading (SMT) performance impact 17
   2.7. IOR 19
   2.8. European ARM processor based systems 19
      2.8.1. Fulhame (EPCC) 19
3. Processors Intel Skylake 21
   3.1. Architecture 21
      3.1.1. Memory Architecture 22
      3.1.2. Power Management 22
   3.2. Programming Environment 23
      3.2.1. Compilers 23
      3.2.2. Available Numerical Libraries 24
   3.3. Benchmark performance 25
      3.3.1. MareNostrum system 25
      3.3.2. SuperMUC-NG system 28
   3.4. Performance Analysis 33
      3.4.1. Intel Application Performance Snapshot 33
      3.4.2. Scalasca 42
      3.4.3. Arm Forge Reports 44
      3.4.4. PAPI 45
   3.5. Tuning 48
      3.5.1. Compiler Flags 48
      3.5.2. Serial Code Optimisation 50
      3.5.3. Shared Memory Programming-OpenMP 53
      3.5.4. Distributed memory programming -MPI 56
      3.5.5. Environment Variables for Process Pinning OpenMP+MPI 61
   3.6. European SkyLake processor based systems 63
      3.6.1. MareNostrum 4 (BSC) 63
      3.6.2. SuperMUC-NG (LRZ) 67

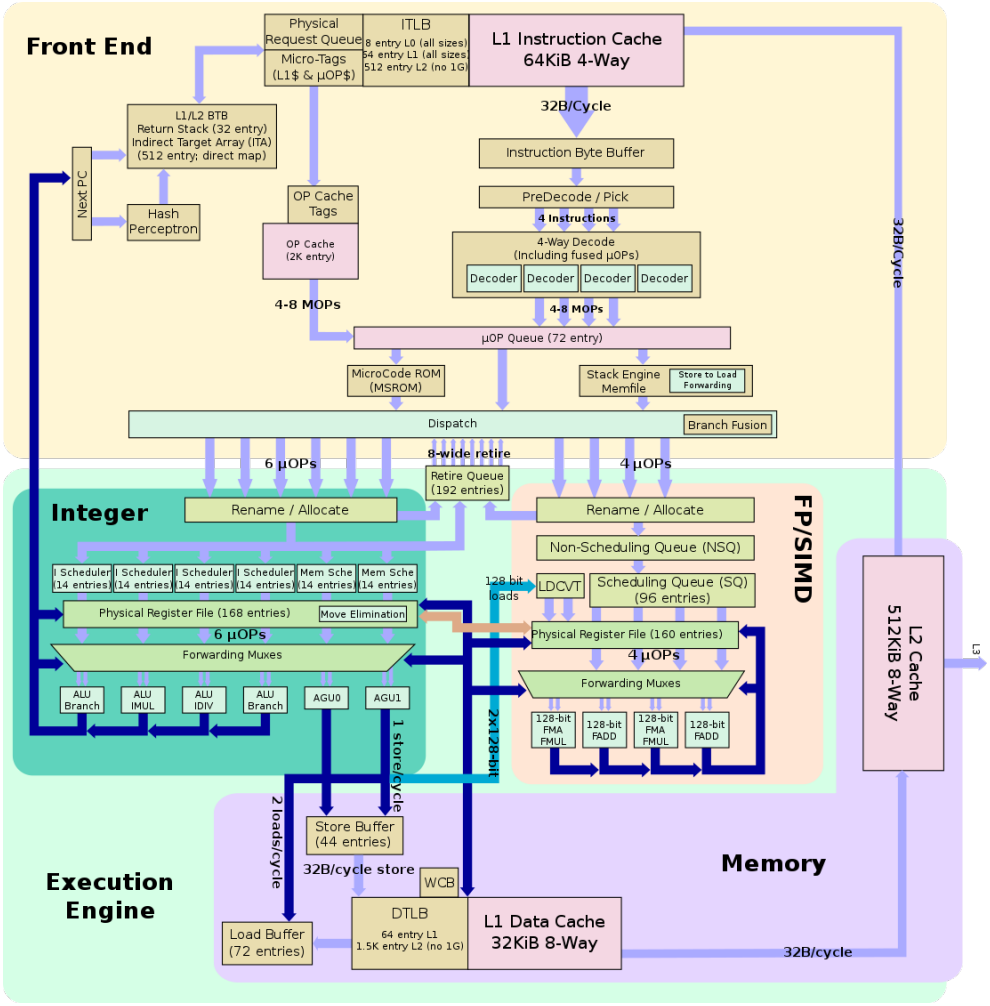
Relevant BPGs – Knights Landing

- High Bandwidth memory
  - Relevant in the future, systems might have HBM2
- Many memory models
  - How do configure HBM2
    - NUMA banks ? - leave the OS & users to deal with it ?
    - Last level cache

Relevant BPGs – x86-64 Intel/AMD

- Haswell/Broadwell, AMD EPYC, Modern Processors (Skylake, Rome)
- Architectures, system design, instruction set
- Memory design
- Compiler and libraries
- Tuning tools (X11 and command line)
- Debuggers (X11 and command line)
- Access to European systems

BPGs – Modern processors – AMD



BPGs – Modern processors

Best Practice Guide
Modern Processors

There are some changes in the cores within the Zen2 architecture. There are four arithmetic logic units (ALUs), and the address generation unit (AGU) count in Zen2 is increased to three. Both ALU and AGU schedulers are improved in Zen2, with increased size for the register file and reordering of the buffers. The imbalance in simultaneous multi threading (SMT) observed in Naples ALUs and AGUs is dealt with by tweaking the algorithms.

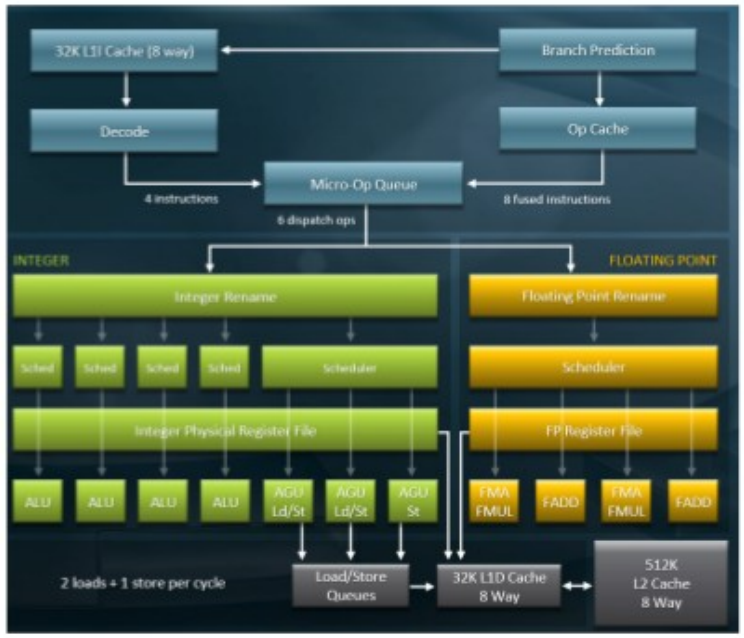
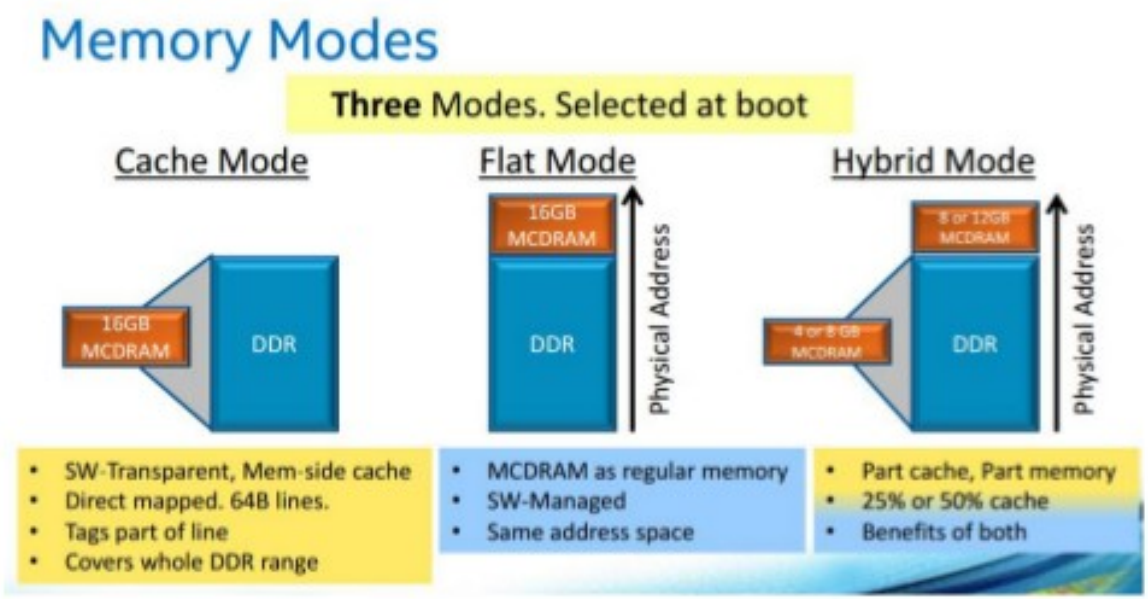


Figure 54. Integer and floating-point instruction units in the Zen 2 cores [46]

BPGs – KNL – HBM / modes

Best Practice Guide
- Knights Landing

Figure 5. Different memory modes"

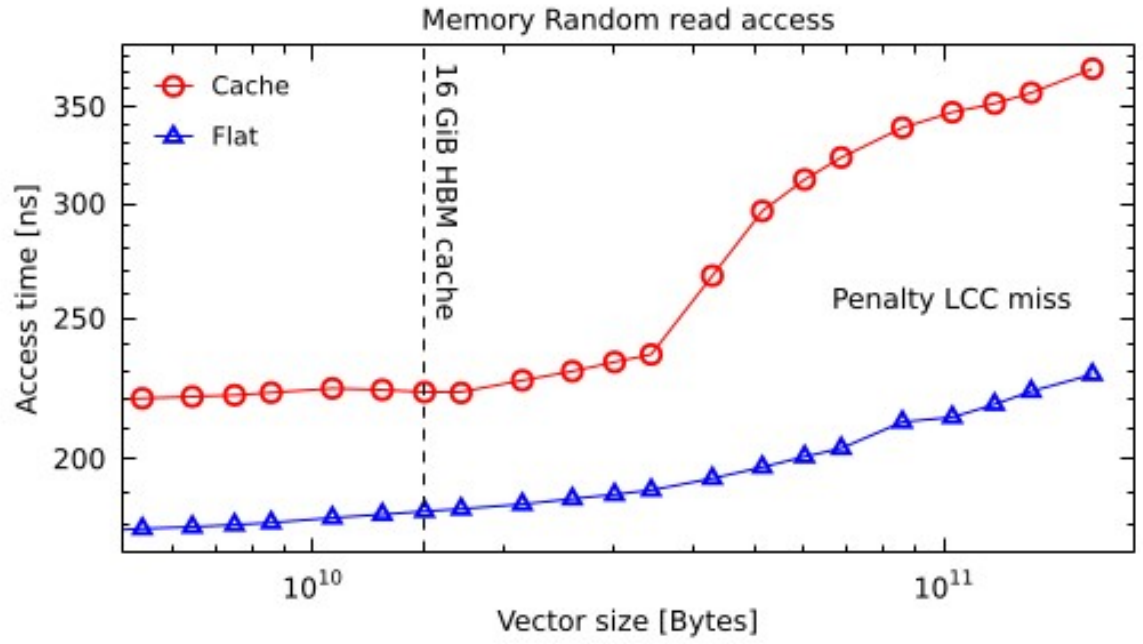


As can be seen in Figure 5, the KNL memory can work in three different modes. These are determined by the BIOS at POST time and thus require a reboot to switch between them.

BPG – KNL – HBM as cache penalty

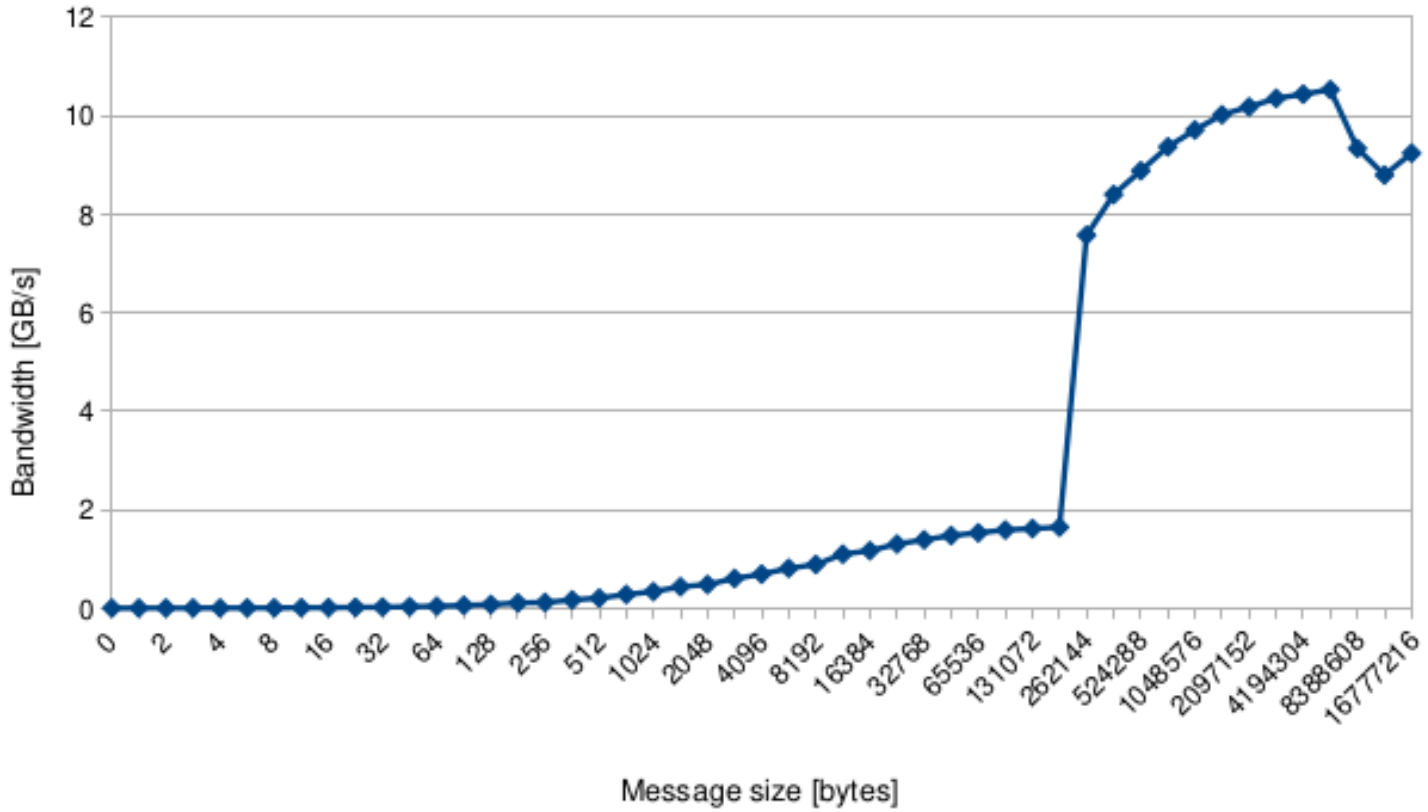
Best Practice Guide
- Knights Landing

Figure 7. Random read memory access for large vectors



BPGs – ARM – Interconnect RoCE

The MPI ping pong latency is measured at about 1.5 microseconds, the bandwidth is approaching wire speed at peak bandwidth with medium to large sized MPI messages.



BPG - Modern processors – AMD

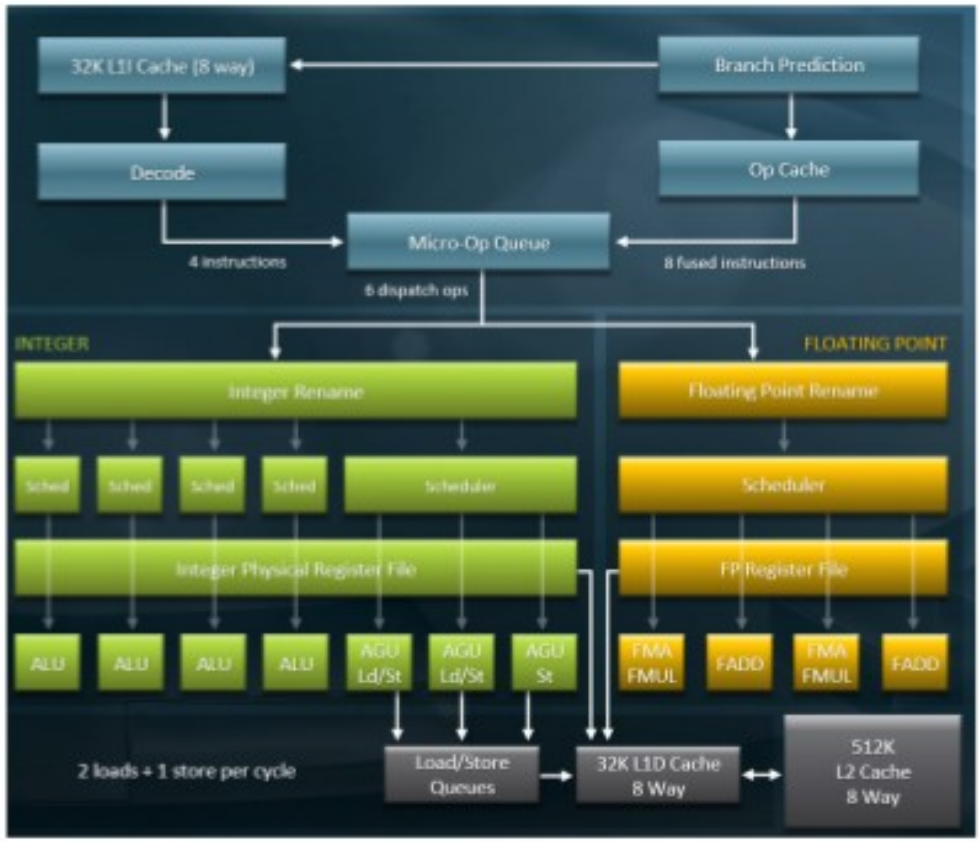


Figure 54. Integer and floating-point instruction units in the Zen 2 cores [46]

BPG – Modern processors – AMD

Best Practice Guide
Modern Processors

4.1.3. NUMA

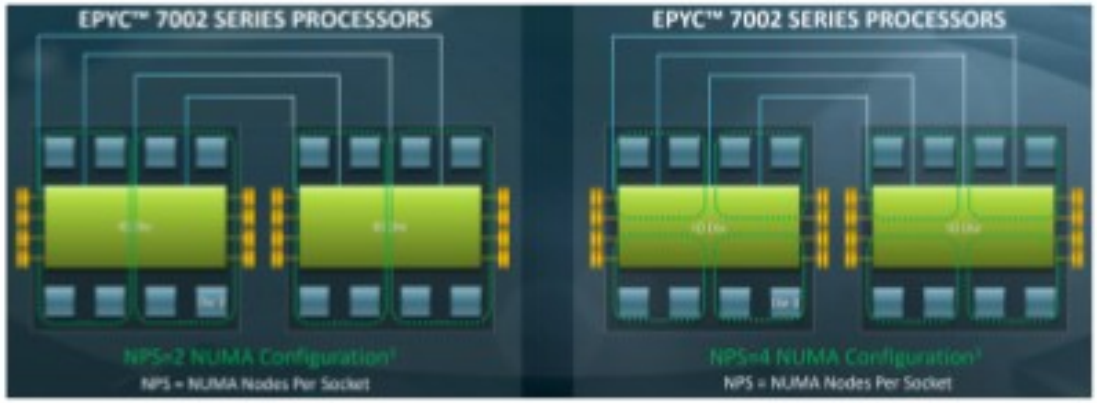


Figure 61. AMD Rome Numa Representations [47]

BPG – Modern processors – AMD

4.2. Programming Environment

4.2.1. Available Compilers

4.2.2. Compiler Flags

4.2.2.1. AOCC

Table 26. Suggested compiler flags for AOCC compilers

Compiler	Suggested flags
clang compiler	-O3 -march=znver2 -mfma -fvectorize -mfma -mavx2 -m3dnow
clang++ compiler	-O3 -march=znver2 -mfma -fvectorize -mfma -mavx2 -m3dnow
flang compiler	-O3 -mavx2 -march=znver2

Table 27. Suggested compiler flags for GNU compilers

Compiler	Suggested flags
gcc compiler	-O3 -march=znver2 -mtune=znver2 -mfma -mavx2 -m3dnow -fomit-frame-pointer
g++ compiler	-O3 -march=znver2 -mtune=znver2 -mfma -mavx2 -m3dnow -fomit-frame-pointer
gfortran compiler	-O3 -march=znver2 -mtune=znver2 -mfma -mavx2 -m3dnow -fomit-frame-pointer

Table 28. Suggested compiler flags for Intel compilers

Compiler	Suggested flags
Intel C compiler	-O3 -march=core-avx2 -fma -ftz -fomit-frame-pointer
Intel C++ compiler	-O3 -march=core-avx2 -fma -ftz -fomit-frame-pointer
Intel Fortran compiler	-O3 -march=core-avx2 -align array64byte -fma -ftz -fomit-frame-pointer

BPG – Modern processors – AMD

Table 29. Performance effect of Vectorisation flags

Vectorisation flag	Single core performance
-O3	4.33 GFLOPS
-O3 -march=core-avx2	4.79 GFLOPS
-O3 -xavx	17.97 GFLOPS
-O3 -xavx2	26.39 GFLOPS
-O3 -xcore-avx2	26.38 GFLOPS

If the main() is compiled with -xAVX2 a test refuse to run in AMD, compile all functions with -xAVX2 and main() with -xcore-avx2. Some test gave better performance using -xAVX2.

BPGs – Modern processors – Intel

Table 32. Performance library performance

Benchmark test	AMD libraries (2.1)	Intel MKL (2018.5)
HPL (128 cores)	3400GFLOPS	3411GFLOPS
FFT 1d (62500MiB) (Single core)	197.428 seconds	103.584 seconds

The Math Kernel Library is a very good option for AMD processors, even if not fully supported.

BPGs – Modern processors – AMD

4.2.4.2. MKL with GNU compiler

When using the GNU compiler the situation is somewhat more complicated. The following lines from the HPL build script might serve as a guideline.

```
-L/opt/intel/mkl/lib/intel64 \
-Wl,--start-group \
/opt/intel/mkl/lib/intel64/libmkl_gf_lp64.a \
/opt/intel/mkl/lib/intel64/libmkl_gnu_thread.a \
/opt/intel/mkl/lib/intel64/libmkl_core.a \
-lmkl_avx2 -Wl,--end-group -lpthread -ldl -lm
```

Using a more simplified pure dynamic linking could look like this:

```
-L/opt/intel/mkl/lib/intel64 -lmkl_gnu_thread\
-lmkl_avx2 -lmkl_core -lmkl_rt
```

BPGs – Modern processors – AMD

Table 36. Performance using Intel MKL with different processor settings

Environment Flag	HPL performance [GFLOPS]
unset MKL_DEBUG_CPU_TYPE	843.16
MKL_DEBUG_CPU_TYPE=5	3217.5

Test with the Intel 2020.1.217 version of the compiler with its associated MKL library no longer honor this flag. While this the case with the version tested, it is not known how widespread this is. Users are advised to run their own tests.

https://documentation.sigma2.no/code_development/building.html#performance-libraries

Next update of the BPG

Research have discovered that MKL call a function called *mkl_serv_intel_cpu_true()* to check the current CPU.The function is simply:

```
int mkl_serv_intel_cpu_true() {
  return 1;
}
```

Compiling this file into a shared library using the following command:

```
gcc -shared -fPIC -o libfakeintel.so fakeintel.c
```

To put the new shared library first in the search path we can use a preload environment variable:

```
export LD_PRELOAD=<path to lib>
```

https://documentation.sigma2.no/code_development/building.html#performance-libraries

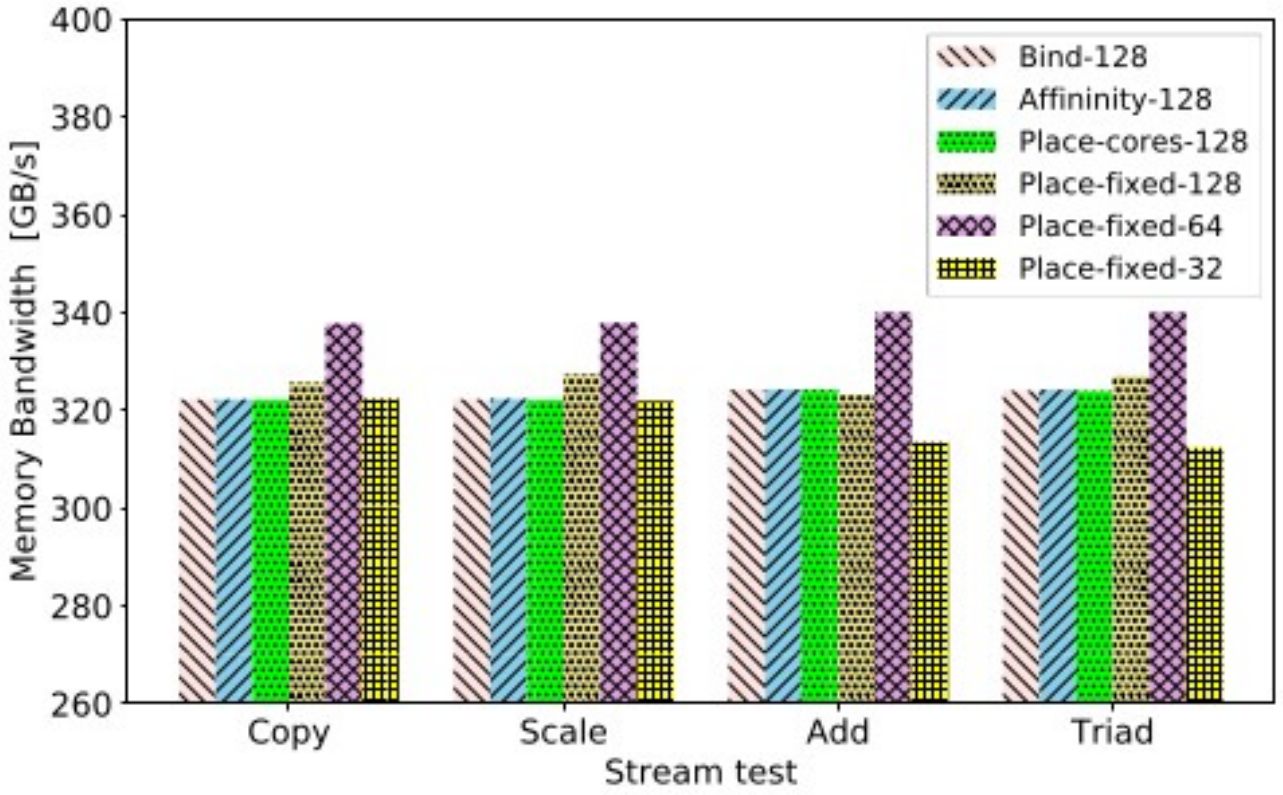
BPGs – Modern processors – Intel

3.2.1.1. Effect of using AVX-512 instructions

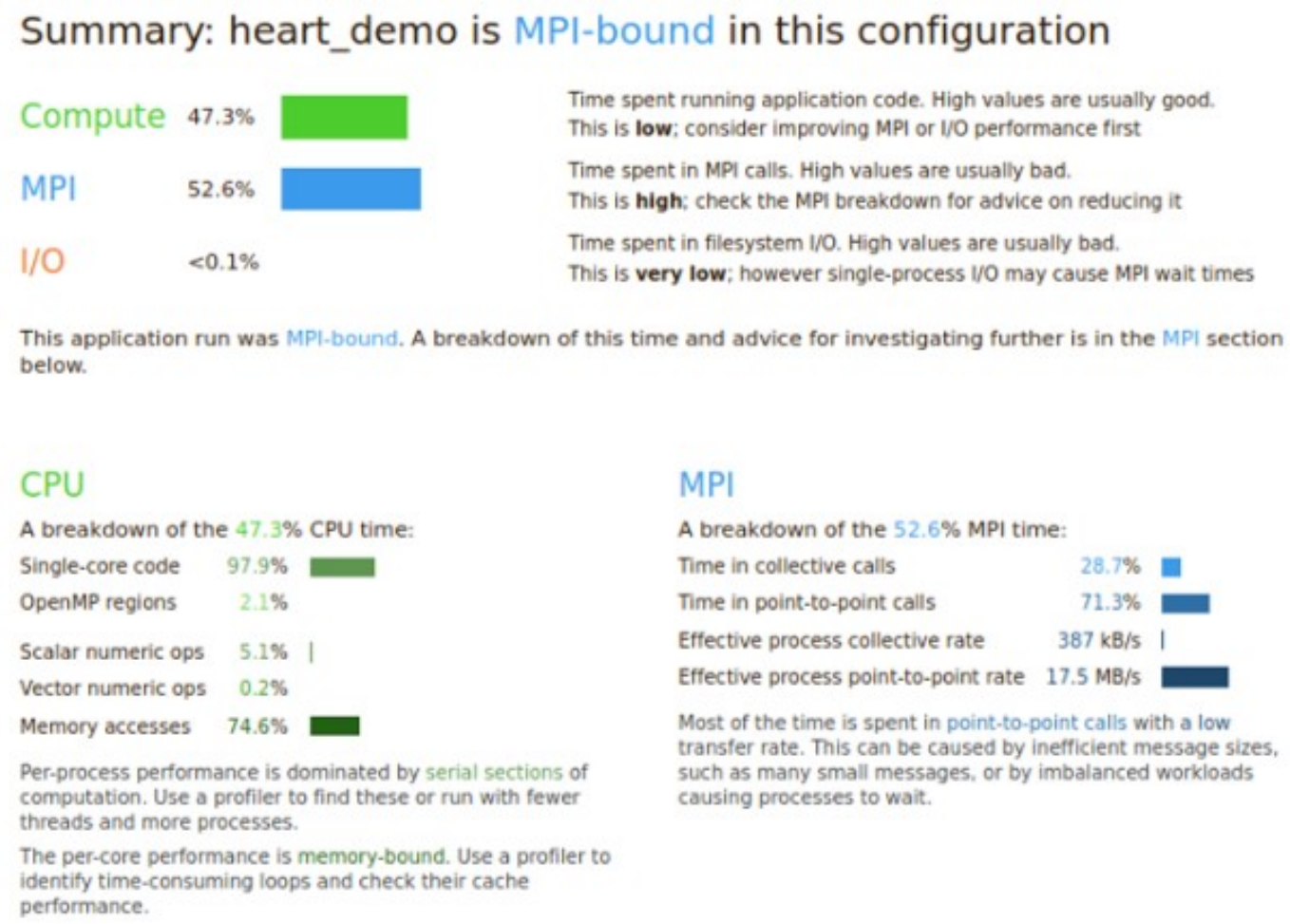
Table 10. HPL Performance with different instruction set

Instruction set	Performance (GFLOPS)	Power (W)	Frequency (GHz)
AVX	1178	768	2.8
AVX2	2034	791	2.5
AVX-512	3259	767	2.1

BPGs – Modern processors – AMD



BPG – ARM perf-reports



Summary: heart_demo is MPI-bound in this configuration

Compute	47.3%	Time spent running application code. High values are usually good. This is **low**; consider improving MPI or I/O performance first
MPI	52.6%	Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it
I/O	<0.1%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was MPI-bound. A breakdown of this time and advice for investigating further is in the MPI section below.

CPU

A breakdown of the 47.3% CPU time:

Single-core code	97.9%
OpenMP regions	2.1%
Scalar numeric ops	5.1%
Vector numeric ops	0.2%
Memory accesses	74.6%

Per-process performance is dominated by serial sections of computation. Use a profiler to find these or run with fewer threads and more processes.

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

MPI

A breakdown of the 52.6% MPI time:

Time in collective calls	28.7%
Time in point-to-point calls	71.3%
Effective process collective rate	387 kB/s
Effective process point-to-point rate	17.5 MB/s

Most of the time is spent in point-to-point calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

BPGs – Intel tuning tools

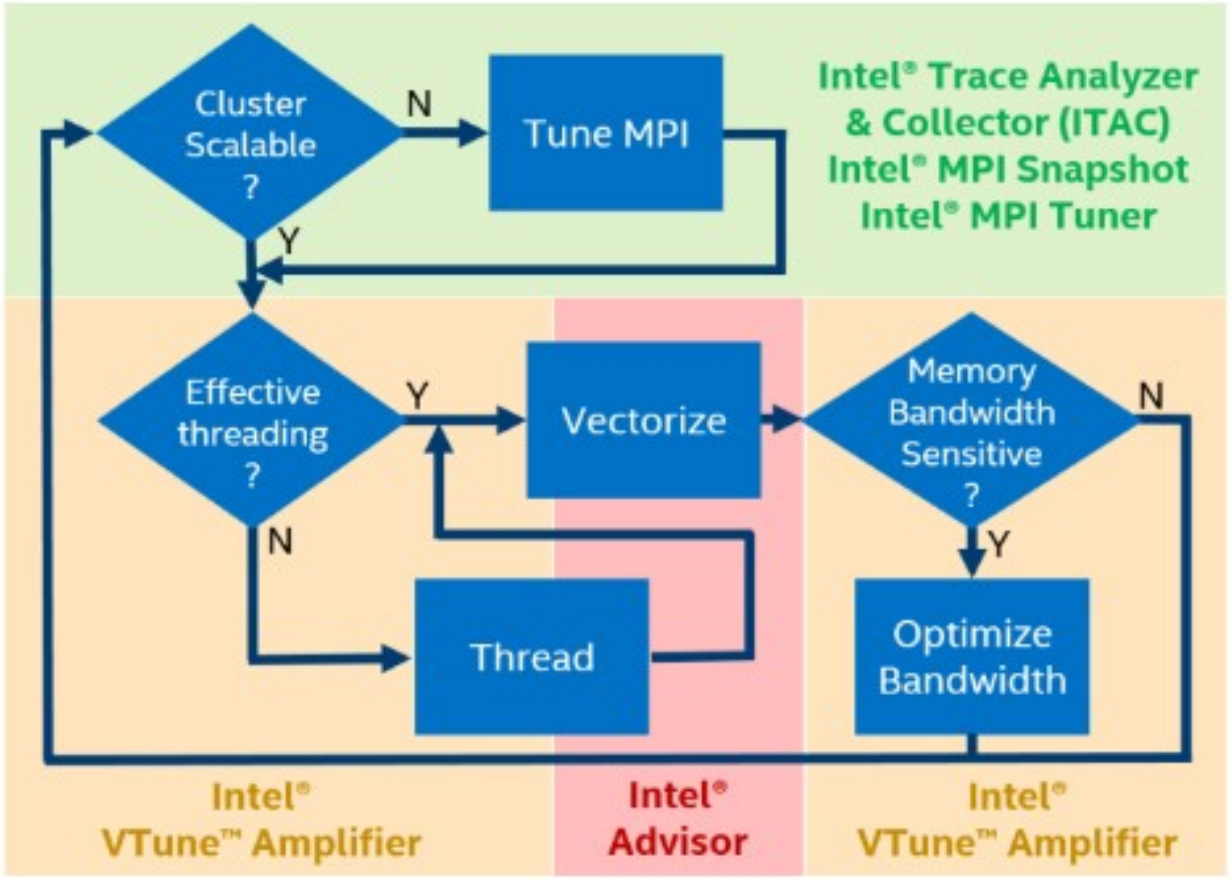


Figure 26. Intel profiling and performance analysis within the Intel Parallel Studio

BPGs – Intel tuning tools, CLI

```
# compilation
$ icc -qopenmp mat_mat_multiplication.c

# code execution
$ aps --collection-mode=all -r report_output ./a.out
$ aps-report -g report_output                    # create a .html file
$ firefox report_output_<postfix>.html           # APS GUI in a browser
$ aps-report report_output                       # command line output


# compilation
$ mpiicc mat_mat_multiplication.c

# code execution
$ mpirun -np 32 aps --collection-mode=mpi -r result_output ./a.out
$ aps-report -g report_output                # create a .html file
$ firefox report_output.html                 # APS GUI in a browser
$ aps-report report_output                   # command line output
```

BPGs – Modern processors – EU systems

2.8. European ARM processor based systems

2.8.1. Fulhame (EPCC)

3.6. European SkyLake processor based systems

3.6.1. MareNostrum 4 (BSC)

3.6.2. SuperMUC-NG (LRZ)

4.6. European AMD processor based systems

4.6.1. HAWK system (HLRS)

4.6.2. Betzy system (Sigma2)

European systems – covered i BPG

THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu