HPC Code Optimization Workshop

Dr. Fabio Baruffa
fabio.baruffa@lrz.de

Dr. Luigi Iapichino
luigi.iapichino@lrz.de
Outline

• Part 1:
  ➢ Introduction and Motivation
  ➢ Modern Computer Architecture
  ➢ Cache and Memory System
  ➢ Roofline model

• Part 2:
  ➢ Optimization Process
  ➢ Nbody Example
  ➢ Enable vectorization: SIMD

• Part 3:
  ➢ Data layout
  ➢ Data alignment
  ➢ Enable OpenMP

• Part 4:
  ➢ Profiling tools
  ➢ Intel® Advisor XE

• Concluding remarks:
  ➢ Intel PCC @ LRZ
  ➢ Intel® MIC Architecture
Profiling tools
Which tool do I use? A roadmap to optimization

We will focus on tools developed by Intel, available to users of the LRZ systems.

Again, we will skip the MPI layer.

VTune is a very rich tool, we will touch it only quickly.

We will dedicate more time (and hands-on) to Advisor.
Profiling with Intel® VTune Amplifier XE

- Powerful tool for analyzing the node-level performance
  - Multiple programming languages (C/C++, Fortran, .NET, Java, Assembly)
  - Support for all latest Intel® processors (incl. Intel® MIC / Broadwell micro-architectures)

- Performance analysis at different levels
  - High-level (code analysis, parallelization efficiency), no special rights needed
  - Low-level (inspection of all architectural components), module driver is required
  - Processor-specific analysis (e.g., utilisation of vector units on Intel® MIC)

- Minimal execution time overhead
  - No recompilation or special linking needed
  - H/W counter sampling and multiplexing → all interesting events gathered once

- Multiplatform (Windows/Linux, 32/64-bit) + complete command-line interface

- Can produce very large traces (~400MB per min. of exec. time)
Hot-spot guided optimization

Typical workflow

1. Compile code with 
   -g -O2 or -g -O3

2. Set the
   environment
   variables or use a
   wrapper script

3. Tweak code input
   for a short
   representative run

VTune
find top hotspots

Optimize
eliminate issues, reduce hotspot
time

Compiler
identify issues in optimization report

Profiling tools
Performance overview

**Performance overview**

- **Wall-clock time**
- **Cumulative CPU time**
- **Performance bottlenecks are highlighted in red**
- **Overall CPU usage**

### Basic Hotspots

#### Elapsed Time: 15.338s

- **CPU Time**
  - **Effective Time**
    - **Idle**: 0s
    - **Spin**: 63.114s
    - **Overhead**: 2.414s
    - **Total Thread Count**: 16
    - **Paused Time**: 0s

### CPU Usage Histogram

- This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time add to the Idle CPU usage value.
Threads behaviour

Function level profiling

Time line of the application

Profiling tools
Source code view

Profiling tools
Closing remarks

The tool is useful and can be used to find:

- Hotspots in the code and possible bottlenecks
- Characterization of the parallelization efficiency
- Possible locks and spinning threads in the application

- More advanced profiling is provided using special kernel modules (memory bandwidth, hardware event-based sampling,...)
- Instrumenting the code for reducing the amount of profiling part in the application
Intel® Advisor XE
Profiling with Intel® Advisor XE

- Modern HPC processors explore different levels of parallelism: between the cores (multi-threading), within a core (vectorization).

- Adapting applications to take advantage of such high parallelism is often defined as code modernization.

- The Intel® Advisor XE is a software tool for vectorization optimization and thread prototyping.

- The tool guides the software developer to resolve issues during the vectorization process.
Vectorization analysis workflow

1. Run Survey
2. Check the Trip-counts
3. Check Dependencies
4. Check Memory Access Patterns
   (Mark-up Loops)

Start
Edit & compile
Take Snapshot
Deeper-dive analysis
5 Steps to efficient vectorization

1. Compiler diagnostics + Performance Data + SIMD efficiency information

2. Guidance: detect problem and recommend how to fix it

3. "Accurate" Trip Counts + FLOPS: understand utilization, parallelism granularity & overheads

4. Loop-Carried Dependency Analysis

5. Memory Access Patterns Analysis

Profiling tools
Creating a new project via GUI

Interface similar to VTune
Setting up the application

Command-line parameters

Environment variables
Hands-on session

- Go to the folder `nbody`
- Take the `base` version, run `make`
- Load the module for Advisor 2017: `module load advisor_xe/2017`
- Run `advixe-gui` for the GUI interface
- Generate the command line, to be executed on your reserved node.
- What is the vectorization performance?
- Try the same with the other different versions of the code (e.g., `ver4`).
- What about the memory access pattern of the top loop?
- Roofline analysis with Advisor.
- Try to recover a complementary view on the optimization steps done during the previous hands-on sessions
Collecting the survey

Generate the command line, and run it on your reserved compute node
Profiling with Intel® Advisor XE

- **How to improve performance**
- **ISA**
- **Hot-spots**
- **What prevents vectorization**
- **Report from the loop**

Profiling tools
Profiling with Intel® Advisor XE

Provide the following points:

- Vectorization informations
- Application intensity
- Number of vector registers

Profiling tools

Traits

Shuffles, Square Roots, Type Conversions, Unpacks

- 20
Profiling tools

Profiling with Intel® Advisor XE

Useful suggestion!

Recommendations to enable vectorization

Issue: Data type conversions present

There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

Recommendation: Use the smallest data type

The source loop contains data types of different widths. To fix: Use the smallest data type that gives the needed precision to use the entire vector register width.

Example: If only 16-bits are needed, using a short rather than an int can make the difference between eight-way or four-way SIMD parallelism, respectively.
Loop optimized (ver4)

Profiling tools

Vectorization efficiency

Loop analytics

Vector length
Memory access pattern (base)

Stride distribution

Profiling tools
Memory access pattern (ver4)

Better stride distribution
Roofline analysis with Advisor - base
Roofline analysis with Advisor - ver4

Performance (GFLOPS)

Self Elapsed Time: 2.558 s  Total Time: 2.558 s

Source Trip Counts: 22

Traits
- Square Roots

Statistics for FLOPS And Data Transfers

GFLOPS 24.2255
AI 1.387

Giga Floating-point Operations Per Second
Per-loop GFLOPS = Total GFLOP / Elapsed Time. Elapsed time is exclusive (self-time-based) wall time from the beginning to the end of loop/function execution. For single-threaded applications Elapsed time is equal to Self-Time.

AI = Arithmetic Intensity - Ratio of Floating-point Operations to L1 Transferred Bytes

Profiling tools
Closing remarks: 6 steps vectorization methodology

1. Measure baseline release build performance: define a metric which makes sense for the code

2. Determine hotspots using Intel® VTune: most-time consuming functions in the application

3. Determine loop candidates using compiler report: 
   -qopt-report=5 -qopt-report-phase=loop,vec

4. Get advise using Intel® Advisor: use the vectorization analysis capability of the tool

5. Implement vectorization recommendations

Profiling tools

more informations: https://software.intel.com/en-us/articles/vectorization-toolkit
Concluding Remarks
The IPCCs are an Intel initiative for code modernization of technical computing codes.

The work primary focus on code optimization increasing parallelism and scalability on multi/many core architectures.

Currently ~70 IPCCs are funded worldwide.

Our target is to prepare the simulation software for new platforms achieving high nodel-level performance and multi-node scalability.

The IPCC @ LRZ / TUM works on four different community codes.
Target of the LRZ team: Gadget

- Leading application for simulating the formation of the cosmological large-scale structure (galaxies and clusters) and of processes at sub-resolution scale (e.g. star formation, metal enrichment).

- Publicly available, cosmological TreePM N-body + SPH code.

- Good scaling performance up to $O(100k)$ Xeon cores (SuperMUC @ LRZ).
Some bits of analysis: VTune profiling

- Severe shared-memory parallelization overhead
- At later iterations, the particle list is locked and unlocked constantly due to the recomputation
- Spinning time 41%

Multi-threading parallelism
Improved performance

- **Lockless scheme**
- **Time spent in spinning only 3%**

Multi-threading parallelism
Vectorization: improvements from IVB to KNL

- Vectorization through localized masking (*if-statement* moved inside the inlined functions).

- Vector efficiency: performance gain / vector length
  - on IVB: 55%
  - on KNC: 42%
  - on KNL: 83%

- Yellow + red bar: kernel workload
- Red bar: target loop for vectorization
Performance comparison: first results including KNL and Broadwell

- Initial vs. optimized including all optimizations for `subfind_density`

- IVB, HSW, BDW: 1 socket w/o hyperthreading.
  KNC: 1 MIC, 240 threads.
  KNL: 1 node, 136 threads.

- Performance gain:
  - Xeon Phi: **13.7x** KNC, **20.1x** KNL.
  - Xeon: **2.6x** IVB, **4.8x** HSW, **4.7x** BDW.

lower is better
Intel® Xeon Phi in a nutshell

1\textsuperscript{st} generation: Knights Corner (KNC)

- 61 cores (@ 1.1 GHz), connected to an Intel Xeon processor - the "host". This is a co-processor!
- Main advantages over the GPU: the programmer can directly login on the card (e.g. ssh) and use standard parallel programming techniques under Linux.
- Due to the large SIMD register-width (512-bit), efficient vectorization of the code is very important for Intel Xeon Phi.

2\textsuperscript{nd} generation: Knights Landing (KNL)

- Up to 72 cores (@ 1.3 GHz), more modern in design, ~ 3x faster than the ones from KNC.
- It is available as bootable processor.
- Overall performance ~ 3TFLOPS (3x KNC).
- Part of the memory available as High Bandwidth Memory.
- Better interconnect among the cores.
- New vector instructions set (AVX512), compatible with Xeon.
36 Tiles interconnected by 2D Mesh
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384GB
IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset
Node: 1-Socket only
Fabric: Omni-Path on-package (not shown)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

From presentation: Avinash Sodani, Intel Corp.