Performance Methodology & Tools

M. Haefele, M. Lobet

Maison de la Simulation

Acknoledgments: G. Hager, G. Wellein, M. Klemm

Observatoire de Paris, October 8 2018

Outline

Ingredients required for performance evaluation

- Mastering the context of the execution
- Profiling and tracing to find hot spots
- Hardware counters
- Performance models
- Scalability
- Methodology proposition

Context

- Test-case
  - code coverage : physics, solvers . . .
  - characteristic dimensions : problem size with respect to cache size
  - number of occurrence (timestep, iterative method). Impact of functions is relative.
- Compilation environment
- Environment variables
- Process & threasd pinning
- Parallel configuration (running on dedicated resources ?)
- Ensure the application is giving the right answer
- . . .

Profiling and tracing

- A **trace** is a collection of **events** or **timestamps** e
- A **profile** is a collection of **timings** t

$$t_{foo} = \sum_{\#calls} (e_{foo_{out}} - e_{foo_{in}})$$

$\Rightarrow$ $t_{foo} \Leftrightarrow$ **cumulative time** spent in routine foo

Profiles and traces aquisition

- **Instrumentation** (scoreP, gprof): timers and event collectors automatically inserted in the source code
  - Needs to recompile the application
  - Large trace files and large execution time overhead
  - Precise result
- **Sampling** (Vtune, Extrae, EZtrace): application execution is interrupted every $\sim 100\mu s$ and information is stored (call stack, hardware counters...)
  - No need to recompile
  - Smaller trace files and execution time overhead
  - Trace analysis potentially more complicated
  - Sampling rate and test case difficult to tune

Hardware counters

- Pieces of hardware at the core level specific to each processor
- Counts specific events (cache misses, flops, cycles. . .)
- Enriches traces and profilings with additional information

Hot spots

Routines with the largest execution times in a profile

Scalability: definition

Answers the question: if N resources are used instead of 1, is the execution time t divided by N ?

- **Speedup**

$$S(N) = \frac{t(1)}{t(N)}$$

- **Relative efficiency**

$$E(N) = \frac{S(N)}{N} = \frac{t(1)}{Nt(N)}$$

- $S(N) \sim N$ or $E(N) \sim 100\% \Rightarrow$ Application scales
- $S(N) < N/2$ or $E(N) < 50\% \Rightarrow$ Application does not scale

Scalability: Amdhal's law

- Serial α_s and parallel α_p fractions of the **source code**

$$t(1) = (\alpha_s + \alpha_p)t(1)$$

- Assuming a **perfect scaling** of the parallel fraction

$$t(N) = (\alpha_s + \alpha_p/N)t(1)$$

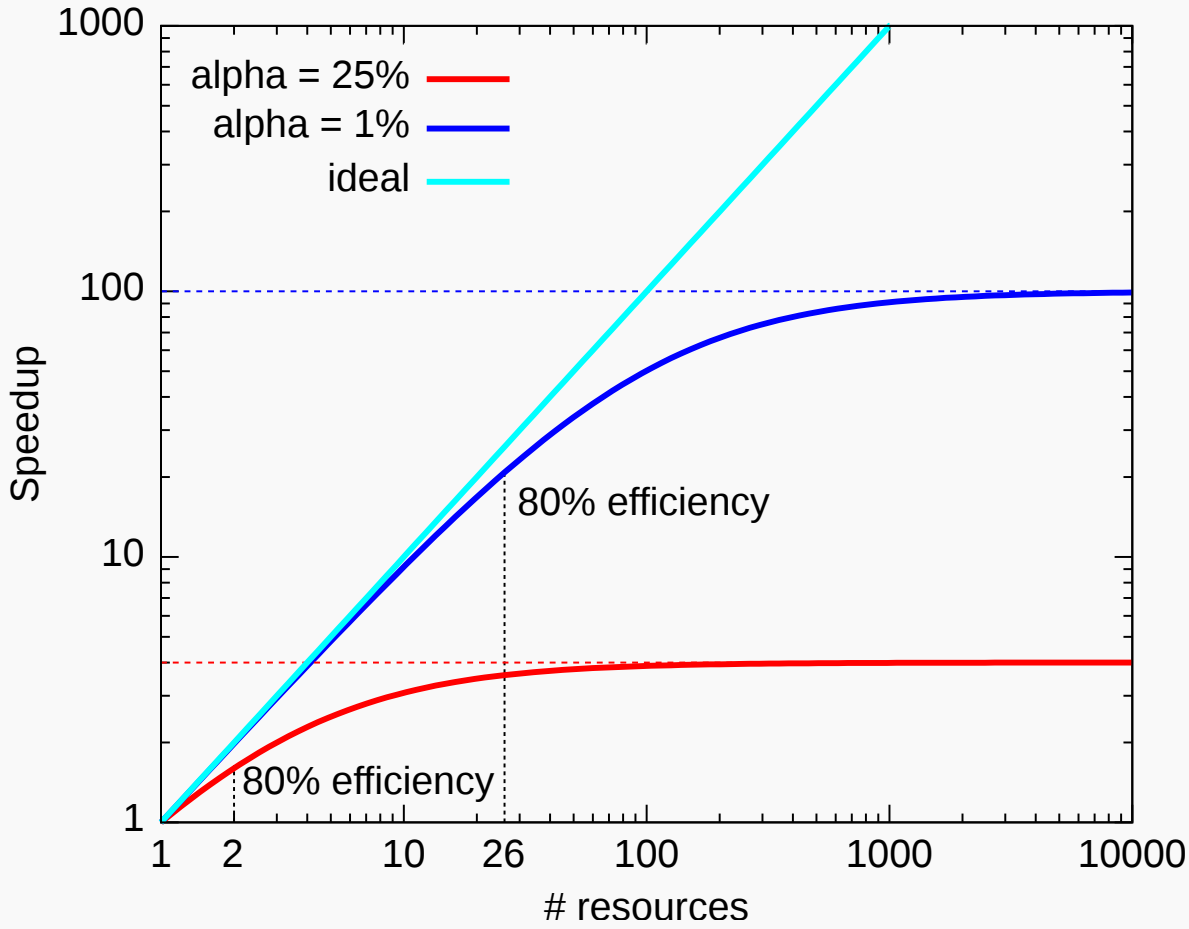
- The speedup reads

$$S(N) = \frac{t(1)}{t(N)} = \frac{1}{\alpha_s + \alpha_p/N}$$

- Assuming a **perfect and unlimited parallel computer**

$$\lim_{N\to\infty} S(N) = \frac{1}{\alpha_s}$$

Scalability: Amdhal's law



Scalability: Strong vs Weak scaling

- **Strong scaling**: same global problem size when resources $\nearrow$

$\Rightarrow$ problem size per resource $\searrow$ when resources $\nearrow$

- **Weak scaling**: global problem size $\nearrow$ with resources

$\Rightarrow$ same problem size per resource when resources $\nearrow$

Scalability: Potential issues

- Load imbalance
- Parallelization overhead
- Number and size of communications

Scalability: presentation

- Present **performance oriented metrics** rather than speedups
  - GFlop/s
  - Simulated time / seconds of simulation time
  - Number of convergence iterations / seconds of simulation time

⇒ Enable to exhibit single core or single node optimization

- Separate intra-node from inter-node scalability

Methodology

1. **Single core optimisation**
   - Find hotspots and measure performance
   - Code improvement (vectorisation, memory access, . . .)
2. **Single node/socket optimisation**
   - Find hotspots and measure OpenMP overhead / imbalance
   - Code improvement (OpenMP, NUMA, . . .)
3. **Inter nodes optimisation**
   - Find hotspots and measure MPI overhead / imbalance
   - Code improvement (Com pattern, parall. overhead, . . .)

Outline: Tools

Tools for measuring performance

- Some existing tools
- Tools used during this training

A very large ecosystem

Compiler

- Vectorization report
- Optimization report

Tools based on instrumentation

- gprof (with some sampling. . .)
- Scalasca/ScoreP
- Tau
- ITAC

A very large ecosystem

Tools based on sampling

- Extrae, Dimemas
- VTune, IPM, advisor
- Inspector, threadspotter
- EZtrace
- Allinea Opt, Map, perf report
- Darshan

A very large ecosystem

Visualization

- Vampir (ScoreP trace)
- Vite (EZtrace)
- Cube (Scalasca post-processing)
- Paraver (Extrae trace)

Hardware counter

- Likwid
- PAPI

Others

- MACAO (static analysis)
- IACA/SDE (emulator)