Sources of performance

- We assume a HPC machine to be a collection of interconnected nodes

- Performance depends on:
  - Application and parallelization
    - Application structure
    - Algorithm
    - Data set
    - Programming model
  - Single-node performance
  - Multi-node performance
A pragmatic classification:

**Users:**
- How fast is my application?
  - Execution time
  - Speed-up

**Developers:**
- How close to the absolute best can I be?
  - Estimate peak(s) and distance from peaks

**Budget-holders:**
- How much of my infrastructure am I using?
  - Utilization
Performance “actions”

- Performance measurement
  - Measure execution time
  - Derive metrics such as speed-up, throughput, bandwidth
  - Platform and application implementation are available
  - Data-sets are available
Performance “actions”

- Performance analysis
  - Estimate performance bounds
    - Performance bounds are typically worst-case, best-case, average-case scenarios.
  - Platform & application are real or close-to-real
    - Simplified models
  - Data-sets are models
Performance prediction
  ◦ Estimate application behavior
  ◦ Platform & application are models
  ◦ Data-sets are real
Speed-up

- Serial execution time \( T_S \)
- Parallel execution time \( T_P \)
- Overhead (\( p \) is \# compute units) \( T_O = p \cdot T_P - T_S \)
  - Ideal case: \( T_O = 0 \) (perfect linear speed-up)

Speedup
- Linear: \( S = p \)
- Sublinear: \( S < p \)
- Superlinear: \( S > p \)

Is superlinear speed-up possible in theory?
In practice: cache effects + problem decomposition.
Superlinear speed-up: example

- $T_{\text{cache}} = 2$ ns, $T_{\text{mem}} = 100$ ns, $T_{\text{penalty}} = 0$
- Problem: $k$ FLOPs, 1 FLOP/mem access
- With 1 Core: Cache hit = 80%.
  \[ T_{\text{FLOP1}} = 2 \times 0.8 + 100 \times 0.2 = 21.6 \text{ ns} \]
- With 2 Cores: Cache hit rate improves to 90%
  \[ T_{\text{FLOP2}} = 2 \times 0.9 + 100 \times 0.1 = 11.8 \text{ ns} \]

\[ S = k \times T_{\text{FLOP1}} / (k/2) \times T_{\text{FLOP2}} \approx 3.67 \]
Speed-up

- Always use the best known sequential execution time
- Always vary the input sizes
  - Speed-up can depend on input data
- Always vary the machine “sizes”
  - Scalability is key
Performance estimation

- Amdahl’s law:
  - \( s \) = sequential work
  - \((1-s)\) = parallelizable work
  - \( p \) = number of processors
  - \( S \) = application speedup

\[
S = \frac{T_{seq}}{T_{par}} = \frac{1}{s + \frac{(1 - s)}{p}} \leq \frac{1}{s}
\]

Speedup is bounded by the sequential fraction.
Amdhal’s Law in pictures

Amdahl’s Law

Parallel Portion
- 50%
- 75%
- 90%
- 95%

Speedup

Number of Processors

1  2  4  8  16  32  64  128  256  512  1024  2048  4096  8192  16384  32768  65536
Amdahl’s law in the MC era

- Design multi-cores to address *both* the parallel and sequential parts of the code
  - A lot of cores for high-throughput applications
    - *and*
  - Several cores to improve the performance of the serial part

Problem size: $N$, Processing elements: $P$

$$E(N,P) = \frac{T_{seq}(N)}{T_{par}(N) \times P}$$

- It is a measure of the “goodness” of a parallel solution for a given application.
Scalability

A parallel solution is **scalable** if it maintains its **efficiency** (constant) when increasing P by increasing N.

To translate:
- Can you increase both P and N and keep efficiency the same => scalable application
- Scalability is a measure of the parallelism in the application and its dependence on the problem size.
Simple performance modeling

- Model computation
  - Count number of operations
  - Assume flat memory model

- Model communication
  - Typically simple model, linear with the amount of data items communicated for large volumes of data
  - Only model explicit, distant communication

Assume:
- \( T_s = \text{number_ops} \times t_{op} \)
- \( T_p = \left(\frac{\text{number_ops}}{p}\right) \times t_{op} + T_{\text{comm}} \)
- \( T_{\text{comm}} = \text{number_comm} \times t_{\text{comm}} \)
Practical Performance
Hardware Performance metrics

- Clock frequency [GHz] = absolute hardware speed
  - Memories, CPUs, interconnects
- Operational speed [GFLOPs]
  - How many operations per cycle can the machine do
- Memory bandwidth (BW) [GB/s]
  - Differs a lot between different memories on chip
  - Remember? Slow memory is large, fast memory is small …
- Power [Watt]

- Derived metrics
  - Normalized for comparison purposes …
  - FLOP/Byte, FLOP/Watt, …
Theoretical peak performance

Peak = chips * cores * vectorWidth * FLOPs/cycle * clockFrequency

- Cores = real cores, hardware threads, or ALUs, depending on the architecture

Examples from DAS–4:

- Intel Core i7 CPU = 154 GFLOPs
  - 2 chips * 4 cores * 4-way vectors * 2 FLOPs/cycle * 2.4 GHz
- NVIDIA GTX 580 GPU = 1581 GFLOPs
  - 1 chip * 16 SMs * 32 cores * 2 FLOPs/cycle * 1.544 GHz
- ATI HD 6970 GPU = 2703 GFLOPs
  - 1 chip * 24 SIMD engines * 16 cores * 4-way vectors * 2 FLOPs/cycle * 0.880 GHz
DRAM Memory bandwidth (off-chip)

- Throughput = memory bus frequency * bits per cycle * bus width
  - Memory clock is not the CPU clock (typically lower)
  - Divide by 8 to get GB/s

Examples:
- Intel Core i7 DDR3: \(1.333 \times 2 \times 64 = 21\) GB/s
- NVIDIA GTX 580 GDDR5: \(1.002 \times 4 \times 384 = 192\) GB/s
- ATI HD 6970 GDDR5: \(1.375 \times 4 \times 256 = 176\) GB/s
Memory bandwidths

- On-chip memory can be orders of magnitude faster
  - Registers, shared memory, caches, ...
    - E.g., AMD HD 7970 L1 cache achieves 2 TB/s

- Other memories: depends on the interconnect
  - Intel’s QPI (Quick Path Interconnect): 25.6 GB/s
  - AMD’s HT3 (Hyper Transport 3): 19.2 GB/s
  - Accelerators: PCI-e 2.0: 8.0 GB/s
Power

- Chip manufactures specify Thermal Design Power (TDP)
  - Some definition of maximum power consumption ...
- We can measure dissipated power
  - Whole system
  - Typically (much) lower than TDP
- Power efficiency: FLOPS / Watt
- Examples (with theoretical peak and TDP)
  - Intel Core i7: \( \frac{154}{160} = 1.0 \text{ GFLOPs/W} \)
  - NVIDIA GTX 580: \( \frac{1581}{244} = 6.3 \text{ GFLOPs/W} \)
  - ATI HD 6970: \( \frac{2703}{250} = 10.8 \text{ GFLOPs/W} \)
## Summary

<table>
<thead>
<tr>
<th></th>
<th>Cores</th>
<th>Threads/ALUs</th>
<th>GFLOPS</th>
<th>Bandwidth</th>
<th>FLOPs/Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Niagara 2</td>
<td>8</td>
<td>64</td>
<td>11.2</td>
<td>76</td>
<td>0.1</td>
</tr>
<tr>
<td>IBM BG/P</td>
<td>4</td>
<td>8</td>
<td>13.6</td>
<td>13.6</td>
<td>1.0</td>
</tr>
<tr>
<td>IBM Power 7</td>
<td>8</td>
<td>32</td>
<td>265</td>
<td>68</td>
<td>3.9</td>
</tr>
<tr>
<td>Intel Core i7</td>
<td>4</td>
<td>16</td>
<td>85</td>
<td>25.6</td>
<td>3.3</td>
</tr>
<tr>
<td>AMD Barcelona</td>
<td>4</td>
<td>8</td>
<td>37</td>
<td>21.4</td>
<td>1.7</td>
</tr>
<tr>
<td>AMD Istanbul</td>
<td>6</td>
<td>6</td>
<td>62.4</td>
<td>25.6</td>
<td>2.4</td>
</tr>
<tr>
<td>AMD Magny-Cours</td>
<td>12</td>
<td>12</td>
<td>125</td>
<td>25.6</td>
<td>4.9</td>
</tr>
<tr>
<td>Cell/B.E.</td>
<td>8</td>
<td>8</td>
<td>205</td>
<td>25.6</td>
<td>8.0</td>
</tr>
<tr>
<td>NVIDIA GTX 580</td>
<td>16</td>
<td>512</td>
<td>1581</td>
<td>192</td>
<td>8.2</td>
</tr>
<tr>
<td>NVIDIA GTX 680</td>
<td>8</td>
<td>1536</td>
<td>3090</td>
<td>192</td>
<td>16.1</td>
</tr>
<tr>
<td>AMD HD 6970</td>
<td>384</td>
<td>1536</td>
<td>2703</td>
<td>176</td>
<td>15.4</td>
</tr>
<tr>
<td>AMD HD 7970</td>
<td>32</td>
<td>2048</td>
<td>3789</td>
<td>264</td>
<td>14.4</td>
</tr>
</tbody>
</table>
Absolute hardware performance

Only achieved in the optimal conditions:
- Processing units 100% used
- All parallelism 100% exploited
- All data transfers at maximum bandwidth

But
- No application is like this
- Even difficult to write micro-benchmarks
- Hardware catalogue values are not realistic estimates of upper bounds of performance.
Arithmetic intensity

- The number of arithmetic (floating point) operations per byte of memory that is accessed.
- Ignore “overheads”
  - Loop counters
  - Array index calculations
  - ...

What is the arithmetic intensity of the RGB–gray kernel?

Is this a compute-bound or a memory-bound kernel? Explain why.

Does the kernel type (memory- or compute-bound) dependent on the application, machine, or both?

```c
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        Pixel pixel = RGB[y][x];
        gray[y][x] =
            0.30 * pixel.R
            + 0.59 * pixel.G
            + 0.11 * pixel.B;
    }
}
```
**Compute or memory intensive?**

Arithmetic intensity for several actual many-cores:

<table>
<thead>
<tr>
<th>Device</th>
<th>Arithmetic Intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Niagara 2</td>
<td>1</td>
</tr>
<tr>
<td>IBM bg/p</td>
<td></td>
</tr>
<tr>
<td>IBM Power 7</td>
<td>4</td>
</tr>
<tr>
<td>Intel Core i7</td>
<td>2</td>
</tr>
<tr>
<td>AMD Barcelona</td>
<td>1</td>
</tr>
<tr>
<td>AMD Istanbul</td>
<td>2</td>
</tr>
<tr>
<td>AMD Magny-Cours</td>
<td>3</td>
</tr>
<tr>
<td>Cell/B.E.</td>
<td>8</td>
</tr>
<tr>
<td>NVIDIA GTX 580</td>
<td>12</td>
</tr>
<tr>
<td>NVIDIA GTX 680</td>
<td>15</td>
</tr>
<tr>
<td>AMD HD 6970</td>
<td>17</td>
</tr>
</tbody>
</table>

“A many-core processor is a device built to transform a compute-bound application into a memory-bound one! ”

*Kathy Yelick*
AI for classes of applications

**Arithmetic Intensity**

- $O(1)$
- $O(\log(N))$
- $O(N)$

- SpMV, BLAS1,2
- Stencils (PDEs)
- Lattice Methods
- FFTs
- Dense Linear Algebra (BLAS3)
- Particle Methods

Red = high AI, Yellow = low AI
Attainable performance

- Attainable GFlops/sec
  \[ \text{min ( Peak Floating-Point Performance, Peak Memory Bandwidth } \times \text{ Arithmetic Intensity) } \]

- To translate:
  - If an application is compute-bound => performance is limited by HW peak performance
  - If an application is memory-bound => performance is limited by the load it puts on the memory system
Typical case: application A runs on platform X in $T_{\text{exec}_A}$:

${\text{PeakCompute}}(X) = \text{maxFLOP GLOPS/s}$ \hspace{1cm} (catalogue)
${\text{PeakBW}}(X) = \text{maxBW GB/s}$ \hspace{1cm} (catalogue)
${\text{RooflineCompute}}(A,X) = \text{min}(\text{AI}(A) \times \text{maxBW}, \text{maxFLOP})$ \hspace{1cm} (model)
${\text{AchievedCompute}}(A,X) = \text{FLOPs}(A)/T_{\text{exec}_A}$ \hspace{1cm} (real execution)
${\text{AchievedBW}}(A,X) = \text{MemOPs}(A)/T_{\text{exec}_A}$ \hspace{1cm} (real execution)

$\text{UtilizationCompute} = {\text{AchievedCompute}}(A,X)/{\text{PeakCompute}}(X) < 1$

$\text{UtilizationBW} = {\text{AchievedBW}}(A,X)/{\text{PeakBW}}(X) \geq 1$

${\text{PeakCompute}} \geq {\text{Roofline}} > {\text{AchievedCompute}}$

${\text{PeakBW}} \geq {\text{AchievedBW}}$

Note: $\text{AchievedBW} > \text{PeakBW} \iff \text{faster memories on the chip play a role.}$
Take home message

- Application performance
  - Depends on algorithm, parallelization, programming model, machine
  - May depend on data-set

- Large variety of performance metrics
  - Depend on “end-users”

- Performance numbers
  - Used to understand application behavior – easy
  - Used to benchmark platforms – difficult

- Theoretical performance boundaries
  - Always too optimistic!

- Real performance boundaries
  - Important for understanding efficiency and/or utilization
    Difficult to calculate => estimations