Performance Analysis

Dirk Pleiter

PDC and CST | EECS | KTH

January 2023

Overview

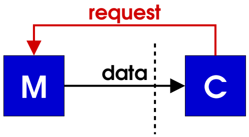
Content

Performance

- Reliable measure of **performance** $= \frac{\text{amount of work}}{\text{execution time}}$
- Execution time = time to execute a particular amount of work
- Ambigiouties in the definition of execution time:
  - Wall-clock time = latency to complete task
  - Resource occupation time, e.g. CPU time

Bandwidth or Throughput vs. Latency

- Definition **bandwidth or throughput**
  - Bandwidth = amount of items passing a particular interface per time unit
  - Throughput = amount of work done in a given time
- Definition **latency (or response time)** $\Delta t =$ Time between events indicating start of a task/operation and the event signaling its completion
- Example: Memory read operation

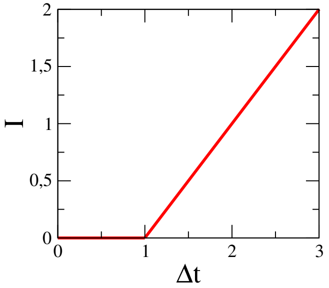


  - Bandwidth: Amount of data passing memory interface per time unit
  - Latency: Time from starting memory read operation until last data item arrived in register file

Latency-Bandwidth Model

- Simple model describing **latency** Δt as a function of the information exchange I
- Model parameters:
  - **Start-up latency** λ = Time between event trigger and start of arrival of response
  - **Bandwidth** β: Parametrizes time to exchange information I
- Model:

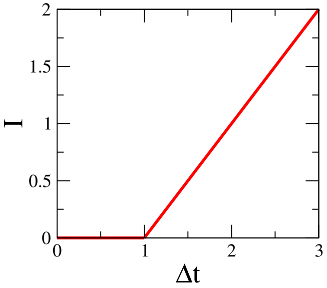
$$\Delta t(I) = \lambda + \frac{I}{\beta}$$

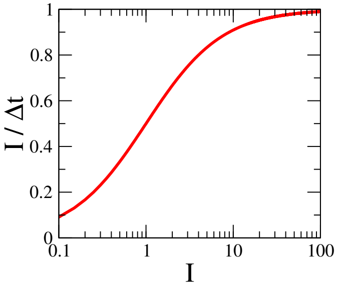


Latency-Bandwidth Model (2)

- **Effective bandwidth** $\tilde{b} = I/\Delta t(I)$
- Example: Read I Bytes from memory or network

$$\Delta t(I) = \lambda + \frac{I}{\beta} \quad \Rightarrow \quad \tilde{b}(I) = \left(\frac{\lambda}{I} + \frac{1}{\beta}\right)^{-1}$$





Information Exchange Function

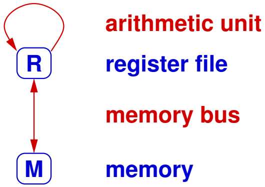
- A computation implies that information is transferred from a storage device x to a storage device y.
- **Information Exchange Function**:

$I^k_{x,y}(W)$ = data transferred between computer subsystems for specific computation k

x	...	source storage device (e.g. memory)
y	...	destination storage device (e.g. register file)
W	...	**problem size/work-load**

Information Exchange Model

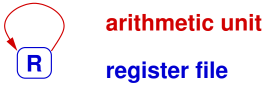
- **Machine** $=$ Set of interconnected devices
  - Storage devices
  - Processing/transport devices
- **Storage devices**
  - Examples: Memory, register file, cache
  - Parameters: Storage size σ
- **Processing/transport devices**
  - Examples: Arithmetic pipeline, bus
  - Parameters: bandwidth/throughput β, startup latency λ
- **Graphical representation**
  - Vertices $=$ Storage devices
  - Edges $=$ Processing/transport devices



Information Exchange Model: Latency Predictions

Ansatz to predict latency:

$$\Delta t^k_{x,y} \simeq \lambda_{x,y} + I^k_{x,y}/\beta_{x,y}$$



Example:

- $x,\ y = \mathrm{R}$
- $\beta_{R,R}$ = throughput arithmetic unit
- $I^k_{x,y}$ = number of input (or output) operands

Beware of limitations of this ansatz:

- Transfer mechanism may depend on task size N
- Bandwidth changes due to resource congestion is ignored
- ...

Information Exchange Model: Simple Example (1/2)

- Example operation: $y_i \leftarrow \alpha \cdot x_i$ $(i = 1, \ldots, N)$

  x_i, y_i ... vectors of single precision floating-point numbers

  α ... single-precision floating-point number
- Information exchange:

load x, store y	$I_\mathrm{mem} = N \cdot (4 + 4)\,\mathrm{Bytes}$
multiply α and x	$I_\mathrm{fp} = N \cdot 1\,\mathrm{Flop}$

- Assume the following hardware parameters:

Memory bandwidth	$\beta_\mathrm{mem} = 1$ Byte/clock cycle
Floating-point unit throughput	$\beta_\mathrm{fp} = 1$ Flop/clock cycle

- Latency predictions (ignoring start-up latency):

Δt_mem	$N \cdot 8\,\mathrm{clock\ cycles}$
Δt_fp	$N \cdot 1\,\mathrm{clock\ cycles}$

Information Exchange Model: Simple Example (2/2)

Latency for full operation:

- No overlap of memory load/store and arithmetic operations:

$$\Delta t(N) = \Delta t_{\mathrm{mem}} + \Delta t_{\mathrm{fp}} = N \cdot 9\,\mathrm{cycle}$$

- Perfect overlap of memory load/store and arithmetic operations:

$$\Delta t(N) = \max\left(\Delta t_{\mathrm{mem}}, \Delta t_{\mathrm{fp}}\right) = N \cdot 8\,\mathrm{cycle}$$

☞ **Memory bandwidth limited problem**

Arithmetic Intensity

- **Arithmetic Intensity** $=$ [H. Harris, 2005]

$$\mathrm{AI} = \frac{\text{Number of floating-point operations}}{\text{Amount of transferred data}} = \frac{I_{\mathrm{fp}}}{I_{\mathrm{mem}}}$$

- Example: $y_i \leftarrow \alpha \cdot x_i$

$$\mathrm{AI} = \frac{1\,\text{Flop}}{8\,\text{Bytes}} \qquad \text{single-precision}$$

$$\mathrm{AI} = \frac{1\,\text{Flop}}{16\,\text{Bytes}} \qquad \text{double-precision}$$

Roofline Model (1/2)

[S. Williams et al., 2009]

- Floating-point and memory performance limits:

$$b_{\mathrm{fp}} \le B_{\mathrm{fp}}, \quad b_{\mathrm{mem}} \le B_{\mathrm{mem}}$$

- Upper limit for latency assuming perfect overlap of memory and arithmetic operations assuming the latency-bandwidth model to hold with start-up latencies:

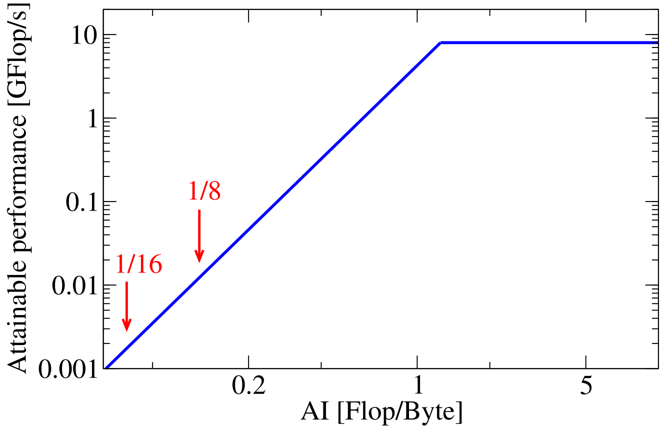
$$\Delta t = \max\left(\frac{I_{\mathrm{fp}}}{b_{\mathrm{fp}}}, \frac{I_{\mathrm{mem}}}{b_{\mathrm{mem}}}\right) \ge \max\left(\frac{I_{\mathrm{fp}}}{B_{\mathrm{fp}}}, \frac{I_{\mathrm{mem}}}{B_{\mathrm{mem}}}\right)$$

- Upper limit (=roof) for floating-point performance ("attainable performance")

$$b_{\mathrm{fp}} = \frac{I_{\mathrm{fp}}}{\Delta t} \le \min\left(B_{\mathrm{fp}}, \frac{I_{\mathrm{fp}}}{I_{\mathrm{mem}}} B_{\mathrm{mem}}\right) = \min\left(B_{\mathrm{fp}}, \mathrm{AI} \cdot B_{\mathrm{mem}}\right)$$

Roofline Model (2/2)

Example: $y_i \leftarrow \alpha \cdot x_i$



Content

Performance measurement

- Types
  - **Profiling**
    Characterize behaviour of an application in terms of aggregate performance metrics
    - Example: Average time spent in subroutine
  - **Sampling**
    Creation of profies by means of period or random probing of performance numbers independently of the current state of application execution.
  - **Event-tracing**
    Generate list of event records comprising of time stamp, location, type information, event-specific information
- Typically additional instructions need to be inserted into program ("instrumentation")

Performance Measurement (2)

- **Hardware support for performance measurement**
  - Modern processors provide hardware counters organised in a Performance Monitoring Unit (PMU)

    Hardware counters = Small set of registers that count events
  - Typical events
    - Clock tick
    - Cache miss
    - Instruction scheduled
- **Issues related to performance measurement**
  - Code instrumentation may impact performance
  - Accessibility of hardware counters
    - Typically support from kernel is required
    - System-wide services may be used for collecting data
  - Management of large event traces

Typical Time Scales

Example: Intel Nehalem architecture

Core clock cycle	$300 - 500\,\mathrm{ps}$
L1 cache access latency	$\sim 5\,\mathrm{cycles}$
L2 cache access latency	$\sim 10\,\mathrm{cycles}$
L3 cache access latency	$\sim 30 - 50\,\mathrm{cycles}$
Memory access latency	$200 - 300\,\mathrm{cycles}$

☞ Need time resolution $\ll 1\,\mu\mathrm{sec}$

Time Measurement

- `/usr/bin/time`
  - Wall, user, system time
  - Resolution given by ticks per second:
    `sysconf(_SC_CLK_TCK)` typically: 10 ms
- `clock_gettime(clockid_t, struct timespec *)`
  - Support for different clocks, recommended to use
    `CLOCK_PROCESS_CPUTIME_ID` or `CLOCK_THREAD_CPUTIME_ID`
  - Returns current clock as structure

```
struct timespec {
    time_t tv_sec;    /* seconds */
    long tv_nsec;     /* nanoseconds */
};
```

  - Include file `time.h` and link with `-lrt`

Performance Tools: PAPI

PAPI = Performance Application Programming Interface

- Developed at University of Tennessee
- De facto standard to interface with PMU on various architectures
  - Examples: Intel Xeon, Xeon Phi; AMD; IBM POWER; Arm
- Selected counters supported by PAPI:

`PAPI_TOT_CYC`	Total cycles
`PAPI_TOT_INS`	Instructions completed
`PAPI_FP_INS`	Floating point instructions
`PAPI_L1_DCM`	Level 1 data cache misses
`PAPI_L2_DCM`	Level 2 data cache misses
`PAPI_L3_DCM`	Level 3 data cache misses
`PAPI_BR_INS`	Branch instructions
`PAPI_BR_MSP`	Conditional branch instructions mispredicted

PAPI: Example Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <papi.h>

# define N 10000
double a[N], b[N];

void kernel() {
  for (int i = 0; i < N; i++)
    b[i] = a[i] + 0.1;
}


int main() {
  // Start measurement
  if (PAPI_hl_region_begin("kernel") != PAPI_OK)
    exit(1);

  // Some kernel
  kernel();

  // Read counters
  if (PAPI_hl_region_end("kernel") != PAPI_OK)
    exit(1);

  return 0;
}
```

PAPI: Example Execution

Compile example program:

```
% gcc -o ex.x ex.c -lpapi
```

Execute example program:

```
% export PAPI_EVENTS="PAPI_TOT_CYC,PAPI_LD_INS"
% export PAPI_OUTPUT_DIRECTORY="."
% ./ex.x
```

```
{
  "papi_version":"7.0.0.0",
  "cpu_info":"Hisilicon Kunpeng",
  "max_cpu_rate_mhz":"2600",
  "min_cpu_rate_mhz":"200",
  "event_definitions":{
    "PAPI_TOT_CYC":{
      "component":"perf_event",
      "type":"delta"
    },
    "PAPI_LD_INS":{
      "component":"perf_event",
      "type":"delta"
    }
  },
  "threads":{
    "0":{
      "regions":{
        "0":{
          "name":"kernel",
          "parent_region_id":"-1",
          "cycles":"5293",
          "real_time_nsec":"50140",
          "PAPI_TOT_CYC":"106057",
          "PAPI_LD_INS":"61411"
        }
      }
    }
  }
}
```

PAPI: Example Performance Analysis

- For $N = 10000$ loop iterations we measured $N_{\text{ld}} = 61411$ $\Rightarrow n_{\text{ld}} = N_{\text{ld}}/N \simeq 6$
- Inspection of the assembler code: Number of load instructions per loop iteration is $n_{\text{ld}} = 6$
  - Tip: Use `gcc -S ex.c` to obtain the assembler code
- Inspection of the C code: Only need to load `a[i]`, i.e. the optimum would be $n_{\text{ld}} = 1$

```
kernel:
.LFB6:
        .cfi_startproc
        sub     sp, sp, #16
        .cfi_def_cfa_offset 16
        str     wzr, [sp, 12]
        b       .L2
.L3:
        adrp    x0, a
        add     x0, x0, :lo12:a
        ldrsw   x1, [sp, 12]
        ldr     d0, [x0, x1, lsl 3]
        adrp    x0, .LC0
        ldr     d1, [x0, #:lo12:.LC0]
        fadd    d0, d0, d1
        adrp    x0, b
        add     x0, x0, :lo12:b
        ldrsw   x1, [sp, 12]
        str     d0, [x0, x1, lsl 3]
        ldr     w0, [sp, 12]
        add     w0, w0, 1
        str     w0, [sp, 12]
.L2:
        ldr     w1, [sp, 12]
        mov     w0, 9999
        cmp     w1, w0
        ble     .L3
        nop
        nop
        add     sp, sp, 16
        .cfi_def_cfa_offset 0
        ret
```

PAPI: Example Execution (Optimized)

Compile example program with more optimizations enabled:

```
% gcc -O3 -o ex-O3.x ex.c -lpapi
```

Execute example program:

```
% export PAPI_EVENTS="PAPI_TOT_CYC,PAPI_LD_INS"
% export PAPI_OUTPUT_DIRECTORY="."
% ./ex-O3.x
```

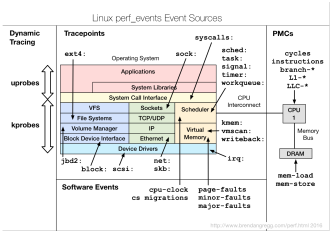
```
{
  "papi_version":"7.0.0.0",
  "cpu_info":"Hisilicon Kunpeng",
  "max_cpu_rate_mhz":"2600",
  "min_cpu_rate_mhz":"200",
  "event_definitions":{
    "PAPI_TOT_CYC":{
      "component":"perf_event",
      "type":"delta"
    },
    "PAPI_LD_INS":{
      "component":"perf_event",
      "type":"delta"
    }
  },
  "threads":{
    "0":{
      "regions":{
        "0":{
          "name":"kernel",
          "parent_region_id":"-1",
          "cycles":"2500",
          "real_time_nsec":"22840",
          "PAPI_TOT_CYC":"40591",
          "PAPI_LD_INS":"6297"
        }
      }
    }
  }
}
```

PAPI: Example Performance Analysis (Optimized)

- For $N = 10000$ loop iterations we measured $N_{\mathrm{ld}} = 6297$ $\Rightarrow n_{\mathrm{ld}} = N_{\mathrm{ld}}/N \simeq 0.5$
- Inspection of the assembler code: Number of load instructions per loop iteration is $n_{\mathrm{ld}} = 1$
- Question: Why do we observe less load instructions than expected?

```
kernel:
.LFB22:
        .cfi_startproc
        adrp    x0, .LC0
        adrp    x2, b
        adrp    x1, a
        mov     x3, 14464
        add     x2, x2, :lo12:b
        add     x1, x1, :lo12:a
        ldr     q1, [x0, #:lo12:.LC0]
        movk    x3, 0x1, lsl 16
        mov     x0, 0
        .p2align 3,,7
.L2:
        ldr     q0, [x1, x0]
        fadd    v0.2d, v0.2d, v1.2d
        str     q0, [x2, x0]
        add     x0, x0, 16
        cmp     x0, x3
        bne     .L2
        ret
```

Performance Tools: Perf



perf = Performance analysis tool for Linux

- Uses the Performance Counters for Linux (PCL)
- Listing of available events: `perf list`
- Example usage:

```
% perf stat ./ex-03.x

 Performance counter stats for './ex-03.x':

            64.14 msec task-clock:u              #    0.731 CPUs utilized
                0      context-switches:u        #    0.000 /sec
                0      cpu-migrations:u          #    0.000 /sec
               59      page-faults:u             #  919.801 /sec
        7,817,710      cycles:u                  #    0.122 GHz
       14,067,551      instructions:u            #    1.80  insn per cycle
    <not supported>      branches:u
           56,321      branch-misses:u

      0.087801579 seconds time elapsed

      0.000000000 seconds user
      0.064650000 seconds sys
```

- Question: Why is execution time much larger compared to PAPI measurements?

Performance Tools: Scalasca

Scalasca

- Developed by FZ Jülich and TU Darmstadt
  - Some functionality became part of a framework developed with other partners
- Open source code that supports various platforms
- Support for different C, C++, and Fortran compilers
- Performance measurements require the following steps:
  1. Tools-based code instrumentation
  2. Performance measurements during code execution
  3. Analysis of performance results

Scalasca: Code Instrumentation

- Tools-based code instrumentation is meanwhile done using Score-P
  - For more information see `https://www.vi-hps.org/projects/score-p/`
- To instrument your code, prefix the compile command with `skin`
- Example:

```
% skin gcc -o ex-nopapi.x ex-nopapi.c
```

Scalasca: Code Execution

- To measure performance, prefix the execution command with `scan`
- Example:

```
% scan ./ex-nopapi.x
S=C=A=N: Scalasca 2.6 runtime summarization
S=C=A=N: ./scorep_ex-nopapi_0_sum experiment archive
S=C=A=N: Sun Jan 15 12:51:16 2023: Collect start
./ex-nopapi.x
S=C=A=N: Sun Jan 15 12:51:17 2023: Collect done (status=0) 1s
S=C=A=N: ./scorep_ex-nopapi_0_sum complete.
```

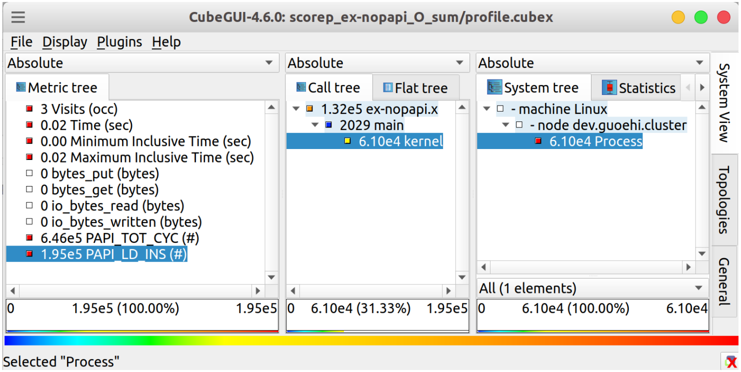
  - Measurement files have been created in `./scorep_ex-nopapi_0_sum`
- To include PAPI counter measurements, define the environment variable `SCOREP_METRIC_PAPI`:

```
% export SCOREP_METRIC_PAPI="PAPI_TOT_CYC,PAPI_LD_INS"
% skin gcc -o ex-nopapi.x ex-nopapi.c
```

Scalasca: Performance Analysis

Options for analysing statistics collected in `profile.cubex`:

- CLI `cube_stat`
- GUI `cube`


CubeGUI-4.6.0: scorep_ex-nopapi_O_sum/profile.cubex

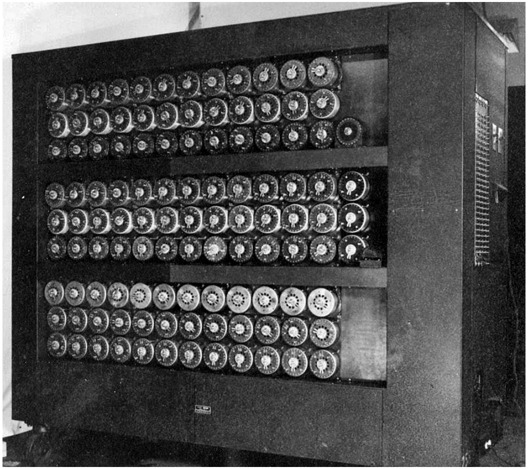
```
3 Visits (occ)
0.02 Time (sec)
0.00 Minimum Inclusive Time (sec)
0.02 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
0 io_bytes_read (bytes)
0 io_bytes_written (bytes)
6.46e5 PAPI_TOT_CYC (#)
1.95e5 PAPI_LD_INS (#)
0        1.95e5 (100.00%)        1.95e5
```

```
1.32e5 ex-nopapi.x
  2029 main
    6.10e4 kernel
0    6.10e4 (31.33%)    1.95e5
```

```
- machine Linux
  - node dev.guoehi.cluster
    6.10e4 Process
All (1 elements)
0        6.10e4 (100.00%)        6.10e4
```

Selected "Process"

Finish with an Architecture from Turing: Bombe



[United Kingdom Government, 1945]