Speedup and Efficiency

Basics of Parallel Computing

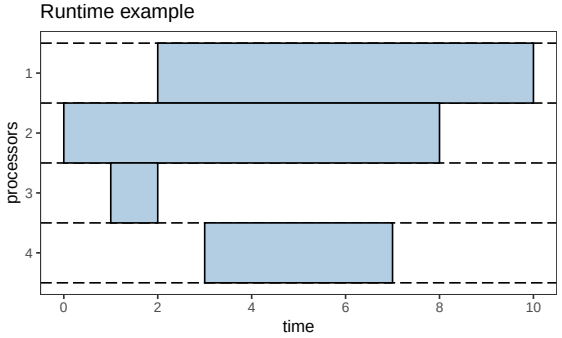
Assoc.Prof. Dr. Sascha Hunold

TU Wien

Running Time

Parallel Runtime [1]

The parallel runtime is the time that elapses from the moment a parallel computation starts to the moment the last processing element finishes execution.



- in this example, processor 2 starts working at time step 0 and 1 ends at times step 10; thus, the running time is 10 time steps

Speed-up I

- number of processors p
- input size n
- parallel running time: $T_{\text{par}}(n, p)$
- sequential running time: $T_{\text{seq}}(n)$

Absolute Speed-up

The absolute speed-up $S_a(n, p)$ is defined as the ratio of the best sequential running time $T_{\text{seq}}(n)$ to the parallel running time on p processors $T_{\text{par}}(n, p)$, i.e.,

$$S_a(n, p) = \frac{T_{\text{seq}}(n)}{T_{\text{par}}(n, p)} \quad .$$

In many cases, we only have access to the parallel program. Or it is possible that we cannot even run a problem on 1 processor. In such cases, we resort to relative speedup metrics.

Speed-up II

Relative Speed-up

The relative speed-up $S_r(n,p)$ is defined as the ratio of the running time $T_{\text{par}}(n,1)$ of the parallel code with 1 processor to the parallel running time on p processors $T_{\text{par}}(n,p)$, i.e.,

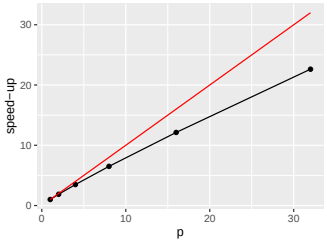
$$S_r(n,p) = \frac{T_{\text{par}}(n,1)}{T_{\text{par}}(n,p)} \quad .$$

Speed-up Example

```
df <- data.frame(p=p, t=partime) # data generated before
df$su <- df[df$p==1,]$t / df$t  # seq time / par time
df <- df  %>% mutate_if(is.numeric, round, digits = 2)
print(toOrg(df))
```

p	t	su
1	1000	1
2	535.89	1.87
4	287.17	3.48
8	153.89	6.5
16	82.47	12.13
32	44.19	22.63

```
p1 <- ggplot(data=df, aes(x=p, y=su)) + geom_point() +
    geom_line() + ylab("speed-up") +
    geom_line(data=df, aes(x=p, y=p), color="red")
plot(p1)
```



- actual speed-up (black)
- ideal/linear speed-up (red)

Parallel Efficiency

- speed-up alone does not always reveal how efficient a parallelization is
- for example there's a difference between
  - a speed-up of 10 with 32 processors
  - a speed-up of 10 with 1024 processors
- that is exactly what the parallel efficiency tells us

Parallel Efficiency

The parallel efficiency is defined as

$$E(n,p) = \frac{T_{\text{seq}}(n)}{p \cdot T_{\text{par}}(n,p)} = \frac{S_a(n,p)}{p} \quad .$$

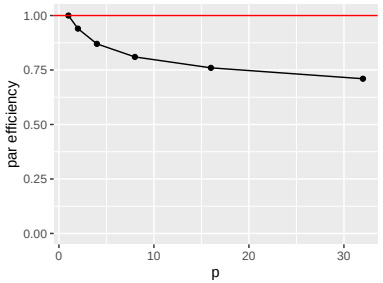
Parallel Efficiency Example

- let us take our speed-up example from before
- for simplicity, we set $T_{\text{seq}}(n) = T_{\text{par}}(n, 1)$

```
df$eff <- df$su / df$p
df <- df  %>% mutate_if(is.numeric, round, digits = 2)
print(toOrg(df))
```

p	t	su	eff
1	1000	1	1
2	535.89	1.87	0.94
4	287.17	3.48	0.87
8	153.89	6.5	0.81
16	82.47	12.13	0.76
32	44.19	22.63	0.71

```
p1 <- ggplot(data=df, aes(x=p, y=eff)) + geom_point() +
    geom_line() + ylab("par efficiency") + ylim(0, 1) +
    geom_hline(yintercept=1, color="red")
plot(p1)
```



Amdahl's Law

Amdahl's Law I

Amdahl's Law

The potential speed-up of a parallel application is bounded by the sequential fraction of the code.

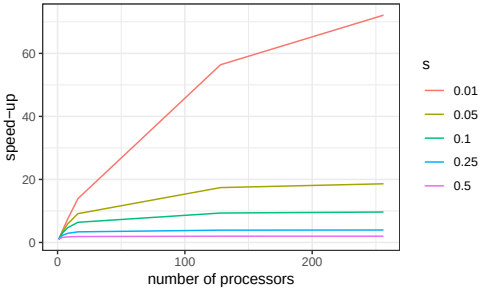
Let $s, 0 \leq s \leq 1$ denote the sequential fraction of a code and $T^*_{\text{seq}}(n)$ the best sequential running time.

Then, the attainable speed-up of the parallel code is

$$S(n,p) = \frac{T^*_{\text{seq}}(n)}{sT^*_{\text{seq}}(n) + \frac{1-s}{p}T^*_{\text{seq}}(n)} = \frac{1}{s + \frac{1-s}{p}} \leq \frac{1}{s} \quad .$$

Amdahl's Law II

```r
library(ggplot2)
su <- function(p, seq_share) {
    1 / (seq_share + (1-seq_share)/p)
}

df <- expand.grid(p = c(1,2,4,8,16,128,256), seq = c(0.5,0.25,0.1,0.05,0.01))
df$su <- su(df$p, df$seq)

ggplot(df, aes(x=p, y=su, color=factor(seq))) + geom_line() +
   xlab("number of processors") + ylab("speed-up") + theme_bw() + labs(color="s")
```



- s is the sequential fraction
- when $p \to \infty$, the maximum achievable speedup becomes $\frac{1}{s}$
- this is what we see

Core Frequency / Possible Speed-up I

- modern processors do not run all cores with the same frequency
- Example: Running Fewer Cores Faster [2]

Active Cores	Max Frequency (GHz)	Breakeven Efficiency
4	2.4	34%
3	2.8	39%
2	3.2	52%
1	3.3	100%

- Breakeven Efficiency: parallel efficiency required to match speed with one core only
- How did they compute this?
  - if we have 1 core with speed 3.3
  - the question is how well we need to use p cores to match 3.3
    - thus, $\frac{f_1}{p \cdot f_p}$
    - for $p = 2$, $\frac{3.3\text{GHz}}{2 \cdot 3.2\text{GHz}} \approx 0.52$

Core Frequency / Possible Speed-up II

- Let us look at a specific processor architecture
- Intel Xeon Gold 6130F Processor (hydra cluster, Q3'17)
  - `https://ark.intel.com/content/www/us/en/ark/products/123688/intel-xeon-gold-6130f-processor-22m-cache-2-10-ghz.html`
  - 16 cores / 32 hyper threads (hyper-threading disabled)
  - Processor Base Frequency: 2.10 GHz
  - Max Turbo Frequency: 3.70 GHz

Core Frequency / Possible Speed-up III

Figure 4. Intel® Xeon® Processor Scalable Family Non Intel® AVX Turbo Frequencies

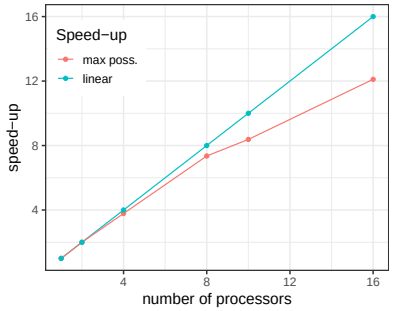
81xx and 61xx processors

					# of active cores / maximum core frequency in turbo mode (GHz)																										
SKU	Cores	LLC (MB)	TDP (W)	Base non-AVX Core Frequency (GHz)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
8176	28	38.50	165	2.1	3.8	3.8	3.6	3.6	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	2.9	2.9	2.9	2.9	2.8	2.8	2.8	2.8
8170	26	35.75	165	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.3	3.3	3.3	3.3	3.0	3.0	3.0	3.0	2.8	2.8	2.8	2.8	2.8	2.8		
8164	26	35.75	150	2.0	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.2	3.2	3.2	3.2	2.9	2.9	2.9	2.9	2.7	2.7	2.7	2.7	2.7	2.7		
8160	24	33.00	150	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.2	3.2	3.2	3.2	3.0	3.0	3.0	3.0	2.8	2.8	2.8	2.8				
6152	22	30.25	140	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	2.9	2.9	2.9	2.9	2.8	2.8						
6138	20	27.50	125	2.0	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.2	3.2	3.2	3.2	2.9	2.9	2.9	2.9	2.7	2.7	2.7	2.7								
6140	18	24.75	140	2.3	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	3.0	3.0										
8153	16	22.00	125	2.0	2.8	2.8	2.6	2.6	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.3	2.3	2.3	2.3												
6130	16	22.00	125	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	2.8	2.8	2.8	2.8												

Core Frequency / Possible Speed-up IV

What does that mean for the possible speed-up?

```
df <- data.frame(p = c(1,2,4,8,10,16),
                 f = c(3.7,3.7,3.5,3.4,3.1,2.8))
# we compute p*f_p / f_1
df$max_su <- df$p*df$f / df[df$p==1,][1,]$f
df$lin_su <- df$p
df2 <- df %>% gather(max_su, lin_su, key="type", value="su")

ggplot(df2, aes(x=p, y=su, color=factor(type))) +
    geom_line() + geom_point() +
    scale_color_discrete(limits = c("max_su", "lin_su"),
                         labels = c("max poss.", "linear")) +
   xlab("number of processors") + ylab("speed-up") +
   theme_bw(base_size=14) + labs(color="Speed-up") +
   theme(legend.position = c(0.15, 0.8))
```



- means: if we actually achieve a speed-up of 12 on this CPU, we are very good (we cannot do better)
- the max. frequency bounds the total achievable speed-up

Relative Speed-up in Practice

Scaling - A Matter of your Reference Point I

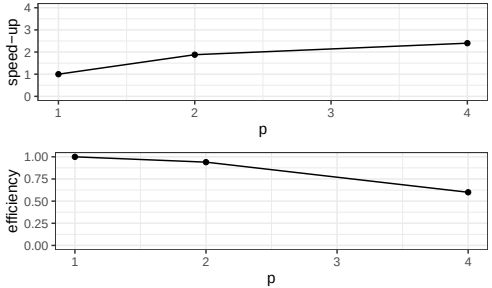
- example: sum up numbers, from 1 to 1, from 1 to 2, from 1 to 3, and so on
- we implement that function in R

```r
library(parallel)
library(rbenchmark)

dummy_count <- function(max) {
  sum = 0
  for(i in 1:max) {
    sum = sum + i
  }
  sum
}

df <- data.frame(p=c(1,2,4), t=c(-1,-1,-1))

for (i in c(1,2,4)) {
  a <- within(benchmark(replications=c(10),
                        mclapply(1:10000, dummy_count, mc.cores = i),
                        columns=c('test', 'elapsed', 'replications')),
              { average = elapsed/replications })
  df[df$p==i, ]$t <- a$average[[1]]
}

write.csv(df, file="./data/runtime.csv", row.names = FALSE)
```

Scaling - A Matter of your Reference Point II

```
df <- read.csv("./data/runtime.csv", header=TRUE)
dfseq <- df %>% filter(p==1)
tseq <- dfseq[1, "t"]
df$su <- tseq / df$t
df$eff <- df$su / df$p
```

p	t	su	eff
1	1.58	1	1
2	0.84	1.88	0.94
4	0.66	2.4	0.6



- since I ran that code on my processor (2 cores, 4 hyperthreads), results not surprising
  - note: the runtimes are in seconds!
- still, we could say that our R code “scales well” with 2 cores
- it does that, but only when considering the relative speed-up

Scaling - A Matter of your Reference Point III

- let us make a better sequential version of that code
- we implement the same function in Julia

```
1 using BenchmarkTools
2
3 function dummy_count(max)
4     sum = 0
5     for i = 1:max
6       sum = sum + i
7     end
8     sum
9 end
10
11 BenchmarkTools.DEFAULT_PARAMETERS.samples = 100000
12 @benchmark (z = map(x -> dummy_count(x), 1:10000))
```

```
BenchmarkTools.Trial:
  memory estimate:  78.20 KiB
  allocs estimate:  2
  --------------
  minimum time:     15.310 µs (0.00% GC)
  median time:      58.758 µs (0.00% GC)
  mean time:        85.041 µs (16.79% GC)
  maximum time:     28.446 ms (99.66% GC)
  --------------
  samples:          58457
  evals/sample:     1
```

- mean time: $85\,\mu s$ (since the slide is generated, the time above may change, but we work in the following with $85\,\mu s$ as the sequential running time)

Scaling - A Matter of your Reference Point IV

- parallel R code was $0.59\,\mathsf{s}$ with 4 cores
  - the relative speedup was 1.78 (which was not even great)
- sequential Julia program solved the problem in $85\,\mu\mathsf{s}$ sequentially
- thus, the absolute speedup is

$$\frac{85 \times 10^{-6}\,\mathsf{s}}{0.59\,\mathsf{s}} \approx 0.00014$$

- thus, we cannot confidently say that our R program scales nicely
- take away message: relative speed-up can be completely misleading

References

[1] A. Grama, G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing*. Second. Addison-Wesley, 2003. ISBN: 0201648652 9780201648652.

[2] M. McCool, J. Reinders, and A. Robison. *Structured Parallel Programming: Patterns for Efficient Computation*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123914439.