Compilers for Embedded Systems

Integrated Systems of Hardware and Software

Lecture 1

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk
Website: https://www.plymouth.ac.uk/staff/vasilios-kelefouras

School of Computing
(University of Plymouth)

Module Outline

Date	Delivery
19th Oct.	Lecture #1
26th Oct.	Lab session #1
2nd Nov.	Lecture #2
9th Nov.	Lab session #2
16th Nov.	Lecture #3
23rd Nov.	Lab session #3
~~30th Nov.~~	Bank holiday
7th Dec.	Lecture #4
14th Dec.	Lab session #4
21st Dec.	Lecture #5
11th Jan.	Lab session #5
18th Jan.	Coursework Support

Module Outline

- **Week1/2.** Motivation, Challenges, Parallel Hardware Architectures, Programming Models and Software Application Profilers, Measuring Performance, Roofline Model
- **Week3/4.**, Memory Hierarchy and Cache, Compiler Options and Code Optimizations - Memory Bound Problems
- **Week5/6**. More advanced Code Optimizations
- **Week7/8**. Vectorization using x86-64 intrinsics
- **Week9/10**. OpenMP Programming – Shared Memory Multi-core CPUs
- **Week11.** Coursework Support

Assessment

- Coursework (100%)

- You will speedup real world applications on your PCs (multi-core CPUs)

Assessment

- Coursework (100%)

- You will speedup real world applications on your PCs (multi-core CPUs)

Outline of the 1st Lecture

- Motivation and Challenges
- Parallel Hardware Architectures and Programming Models
- Software Application Profilers
- How to measure the performance of our parallel software?
- Using timers
- FLOPS
- Speedup, efficiency, scalability
- Roofline Model

High Performance Computing (HPC) market

- The HPC Market Map (next slide) demonstrates the **rapidly growing importance of HPC to industrial competitiveness of both the UK and Europe**
- High performance software is critical in modern computer systems ranging from small embedded devices to big supercomputers and datacenters
- Companies need employers to write efficient software
- Engineers with that knowledge are desirable in Industry

High Performance Computing (HPC)

Special Interest Group

e-infrastructure

8

Introduction

e-Infrastructure was described in a recent report from the Department for Business, Innovation & Skills as the "ecosystem for innovation", underpinning a wide range of future technologies. e-Infrastructure enables computational science & engineering methods and the potential contribution to the UK economy of e-Infrastructure technologies is enormous.

Although higher education currently dominates the UK's HPC resources, enterprises of all sizes are recognising the necessity for HPC in the commercial arena. Few companies are able to make the required investment on their own. Therefore, a need has developed for e-Infrastructure to provide the necessary resources for HPC service providers to meet the rapidly growing demands of UK enterprises.

Trends & Numbers

$10.3 Billion Worldwide revenue value of the HPC Servers market in 2013 (3)

30% European market share of total worldwide HPC revenues in 2013 (3)

$430 Million Value of German Supercomputer segment in 2011or 43% of total HPC revenues (4)

$260 Million Value of French Supercomputer segment in 2011or 44% of total HPC revenues (4)

83% of HPC application software used in Europe is indigineous and 66% is based on IP generated in Europe (2)

<5% European share of HPC system vendors in the global HPC system market (3)

€100 Million Amount committed by PRACE hosting partners (Germany, France, Italy & Spain) to petascale computing cycles between 2010-2014 (1)

Sources

(1) e-Infrastructure Leadership Council 2012
(2) ETP4HPC Strategic Research Agenda 2013
(3) IDC Worldwide High Performance Technical Server Overview 2014
(4) IDC Report on HPC market Trends 2013

Prepared by
FirstPartner
call us on +44 (0) 870 874 8700
find us at www.firstpartner.net

Verticals Served

Energy	Environmental & Geosciences	Defence & Security	Astrophysics	Life Sciences	Chemistry & Materials Science	Automotive / Engineering	Economics & Finance	Supply Chain	Creative & Media
- Advanced combustion modelling - Plasma physics	- Weather forecasting - Climate modelling	- Military vehicle design - Processing of intelligence	- Astrophysical simulation - Telescope data analysis	- DNA sequence analysis - Proteomics and toxicology	- Nanoscale material science - Quantum chemistry	- Simulated prototype testing	- Macro/micro economic modelling - Derivatives trading	- Planning and optimisation of distribution systems	- Computer graphic generation

HPC Users

Research
Public and privately funded research bodies and institutions, typically in nuclear, meteorological, space sciences and biomedical industries

Industry & Business
Large multinational enterprises with the resources to invest in private MPC facilities. Includes banks, automotive manufacturers, aerospace, FMCG, energy, oil and gas industries

SME
Small and medium-sized enterprises in scientific and data-intensive industries, e.g. healthcare, engineering

Educational
Educational institutions, e.g. universities and technical colleges, using HPC facilities to conduct research involving analysis of large datasets, e.g. brain scan data

Government
Governmental departments and agencies, e.g. Department of Defence, as well as other health and regulatory agencies. Governments are key direct and indirect investors in HPC technology

HPC Usage

Big Data Workloads
Manipulating vast datasets that are too large to be processed using commercial software and hardware in short time frames

Simulation & Optimisation
Simulating real-world phenomena to gain deeper understanding or minimise risk, e.g. financial and molecular simulation

Predictive Behaviour
Processing data to create forecast models or predict outcomes, e.g. weather forecasting

Data Informatics
Computationally or numerically intensive processing of vast or complex datasets to aid decision making in a variety of fields

Classification & Textual Analysis
Document or file classification, e.g. music, as well as analyse text, e.g. sentiment analysis

Graphics Generation
Creating computer-generated animation for the Media industry and modelling output representation

Academic

International connectivity: Peering Points ↔ Internet Exchanges; GÉANT

NREN - National connectivity: janet, SURFnet, RENATER, GARR, DFN, JANET, NORDUnet

HPC Access

HPC Resources
Owners
- Commercial
- Public Research
- Academic
- Defense & Security

Access Models
- Pay By Use
- Contracted Use
- Captive In House Use

Commercial

Physical Transfer

Private High Capacity Service

Data Centres

MPC Access Node
- Peering points to NREN
- Direct connection

Point-To-Point Dedicated Connection

Internet Exchanges

Complimentary HPC Approaches

Grid
Distributed computing resources dynamically pooled and built on-top-of high performance networks.
UK Grid Initiative; European Grid Infrastructure; Worldwide LHC Computing Grid

On-Premises
Very large clusters of Servers configured for parallel computing
- Workgroup
- Divisional
- Departmental

Cloud
Cloud services are widely used for application-intensive purposes by commercial organisations.
EC2 HPC; amazon web services; Google; Microsoft Azure

HPC Systems

HPC Software

Application & Development Tools
Vertical sector specific applications & general purpose debugging & optimisation toolkits.

Middleware
Software to manage & optimise multiprocessor clusters to achieve maximum computational performance.

HPC Hardware

Complete HPC Hardware Systems
Top 4 Global HPC System Vendors by Annual Revenue in 2013
IBM $2.8 Bn; $1.5 Bn; $3.3 Bn; $0.4 Bn

European MPC System Vendors

Memory & Storage
New non-volatile forms of memory technology are emerging which can significantly improve energy efficiency and enable the necessary massive throughput and highly parallel access.

Local Interconnects
Communication bandwidth scaling in line with other system components to minimise latency and energy consumption.

Core Processors
High numbers of clusters of cores are used to achieve high throughput in compute-intensive tasks, such as finite element analysis.

System Resilience
The rapidly increasing number of system components requires improving resilience of individual system components to prevent scaling failure.

HPC Scale
Advances in energy efficiency and power over the next decade, driven by iitiatives such as DEEP and Mont Blanc, are key to achieving an exponential growth in computer processing capacity.
Peta — *One quadrillion floating point operations per*
Energy & Power Initatives: IDEEP, MONT BLANC
Exa — *Exponential increase in the number of*

European R&D Resources

Vendors
Commercial HPC system vendors who also undertake original HPC research in Europe.

Institutional
Academic & grant aided centres of research excellence.

Project co-funded by European Commission to build a European vision and roadmap to address the challenges of new generation multi-Petaflop & Exaflop systems.

Existing European Resources

Country		Academic	Research	Industry	Total CPU Cores
	BE	1		1	19,944
	DK			1	15,672
	FI	2			20,976
	FR	5	7	14	776,468
	DE	10	6	6	1,113,012
	IT	1	2	2	268,400
	NL	2		3	62,160
	NO	3			54,400
	ES	1	1		65,280
	SE	2		1	73,248
	CH	2	4		263,888
	UK	5	15	10	925,152

Source: TOP500.org - 06/2014

HPC Initiatives

UK
- UK e-Infrastructure Initiative
- HPC Scotland
- SME Initiatives: A number of initiatives in the UK and Europe include SME access to HPC facilities in their scope.

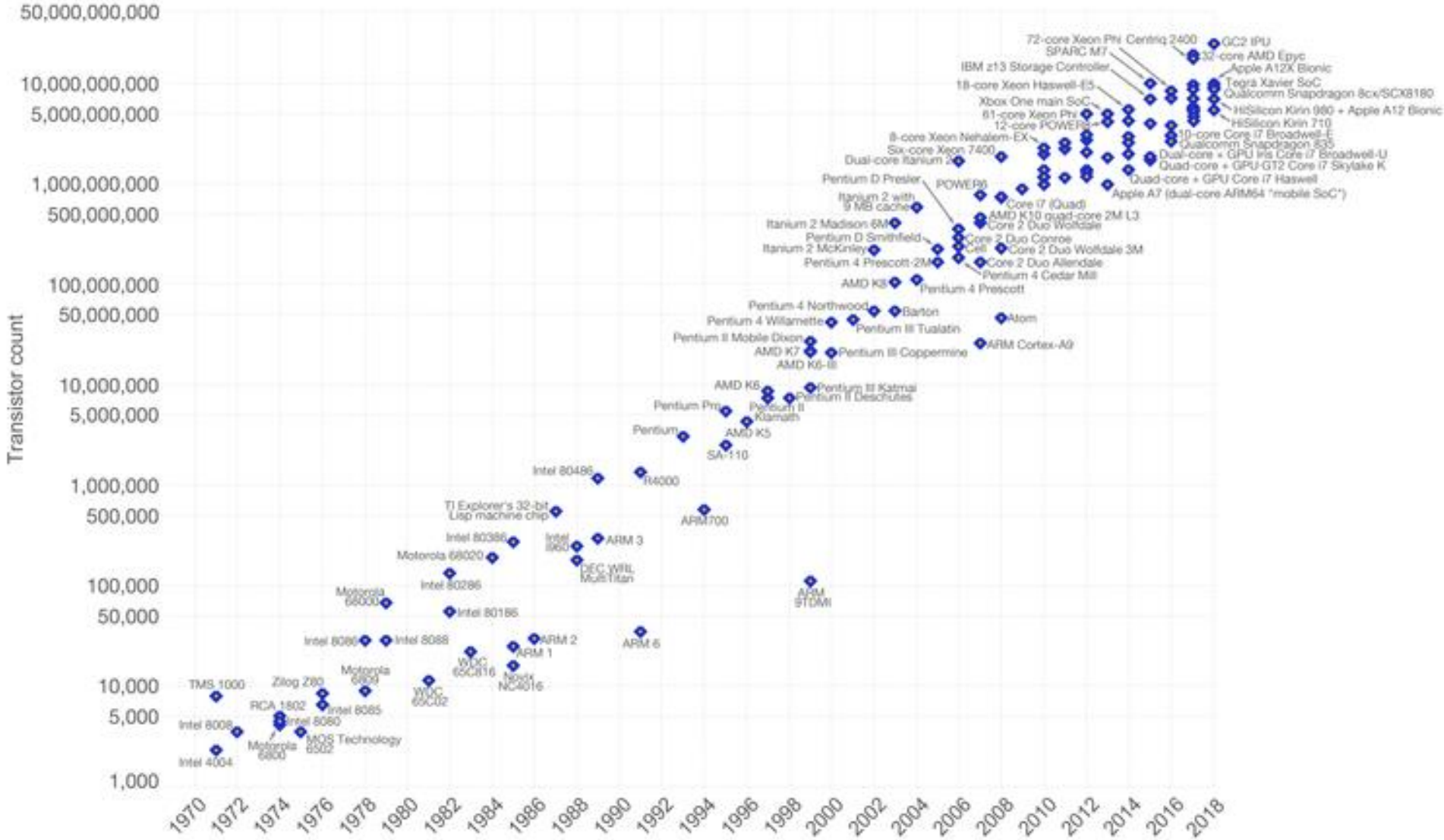
Europe
- SICOS; HLRS; Initiative HPC-PME; Ter@tec
- Network of Excellence on High Performance and Embedded Architecture & Compilation: HiPEAC
- NESUS: Network for Ultrascale Computing
- PRACE: Academic Partnership for Advanced Computing in Europe
- ETP 4 HPC: Industry European Technology Platform 4 HPC
- HPC – Q – 2020
- EESI: European Exascale Software Initiative

[IMPORTANT] - Module Requirements

- Strongly Recommended:
  - **Install a Virtual Machine using Linux** (Ubuntu is recommended)
    - Or even better, native Linux or Windows Subsystem for Linux
- We will be using both Linux and Windows
- Coursework can be done by using either windows or Linux or Mac
  - **If you use Visual Studio, you will need 2019 version** (earlier versions do not work)

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

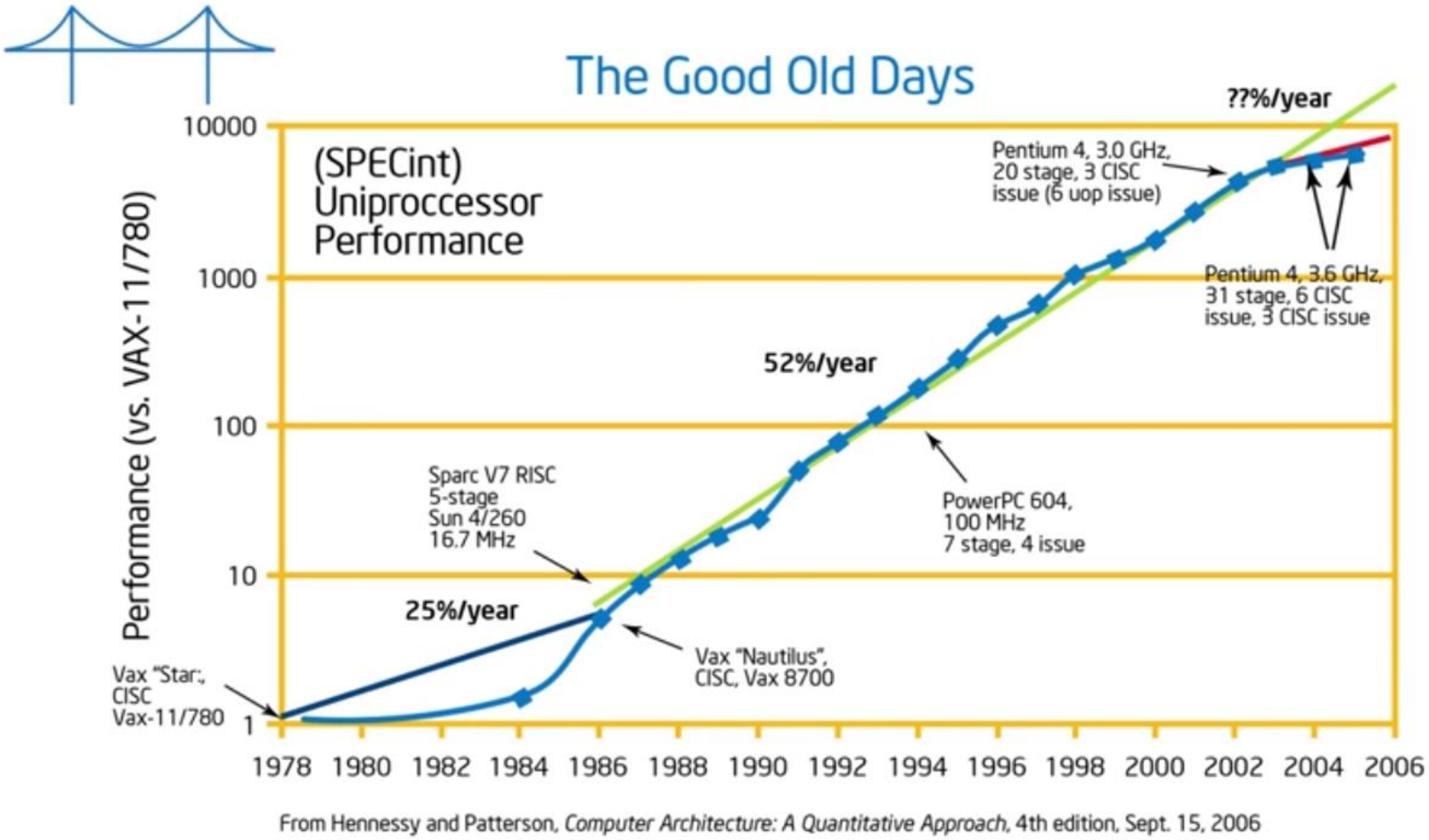
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Performance **used to** increased according to the number of transistors (1)



From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Sept. 15, 2006

Performance **used to** increased according to the number of transistors (2)

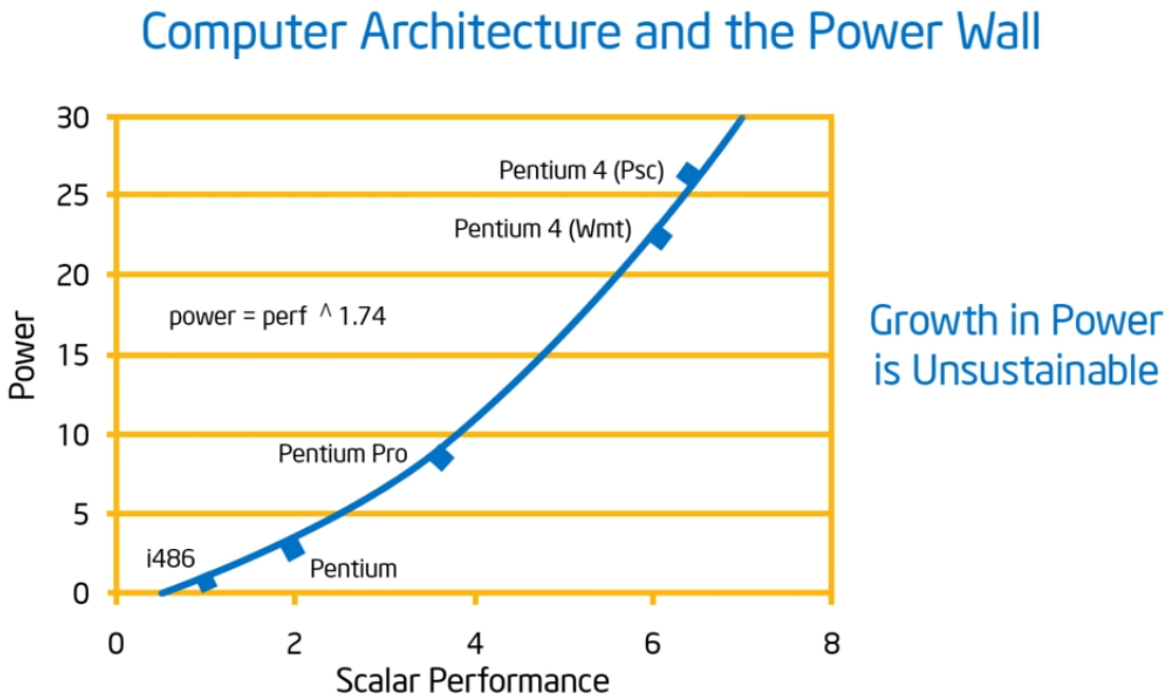
- The fact that performance used to increase by increasing the number of transistors, trained people to expect that performance comes from the hardware
- Programmers used to write software without thinking about performance
- They counted on the hardware to do the work
- This model used to work fine…but…back in 2006, something changed…
  - ***The Power Wall problem***

Performance **used to** increased according to the number of transistors (3)

- Power Wall Problem
  - The CPU design goal for the late 1990's and early 2000's was to increase the CPU frequency.
    - This was a way to improve system performance
    - This was done by adding more transistors to a smaller chip.
    - However, this increased the heat dissipation of the CPU chip beyond the capacity of inexpensive cooling techniques.
    - The last years the ***CPU frequency has ceased to grow***

Performance **used to** increased according to the number of transistors (4)

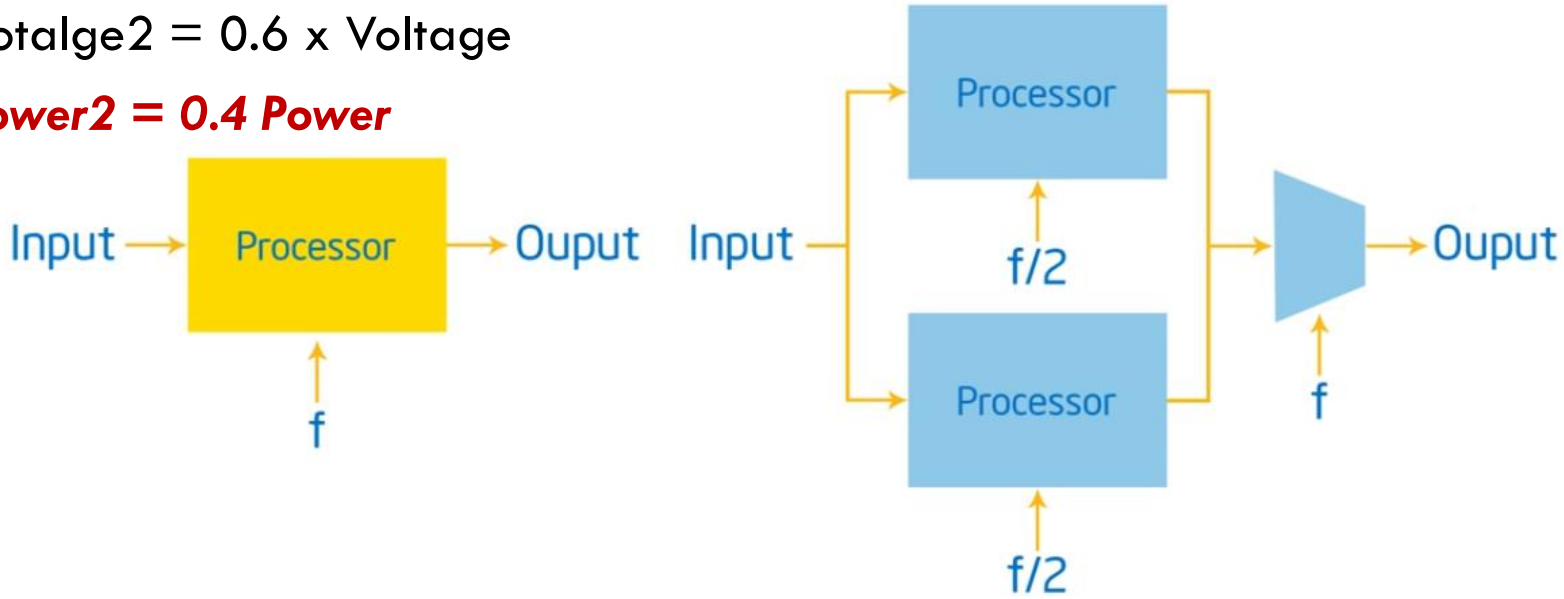
- **By increasing performance, power consumption increases even more** (next slide)
- This is not sustainable
- What to do? ***The solution is Parallel hardware architectures***



Source: E. Grochowski of Intel

The solution to the Power Wall Problem

- The power of a processor is given by **Power=Capacitance x Voltage x Frequency^2**
- By using two processors inside the same chip, with half the frequency each, then:
  - Capacitance2 = 2.2 x Capacitance
  - Frequency2 = F/2
  - Votalge2 = 0.6 x Voltage
  - ***Power2 = 0.4 Power***



Parallel computing gives us the ability to give the same performance with lower power

Chandrakasan, A.P.; Potkonjak, M.; Mehra, R.; Rabaey, J.; Brodersen, R.W., "Optimizing power using transformations," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,, vol.14, no.1, pp.12-31, Jan 1995

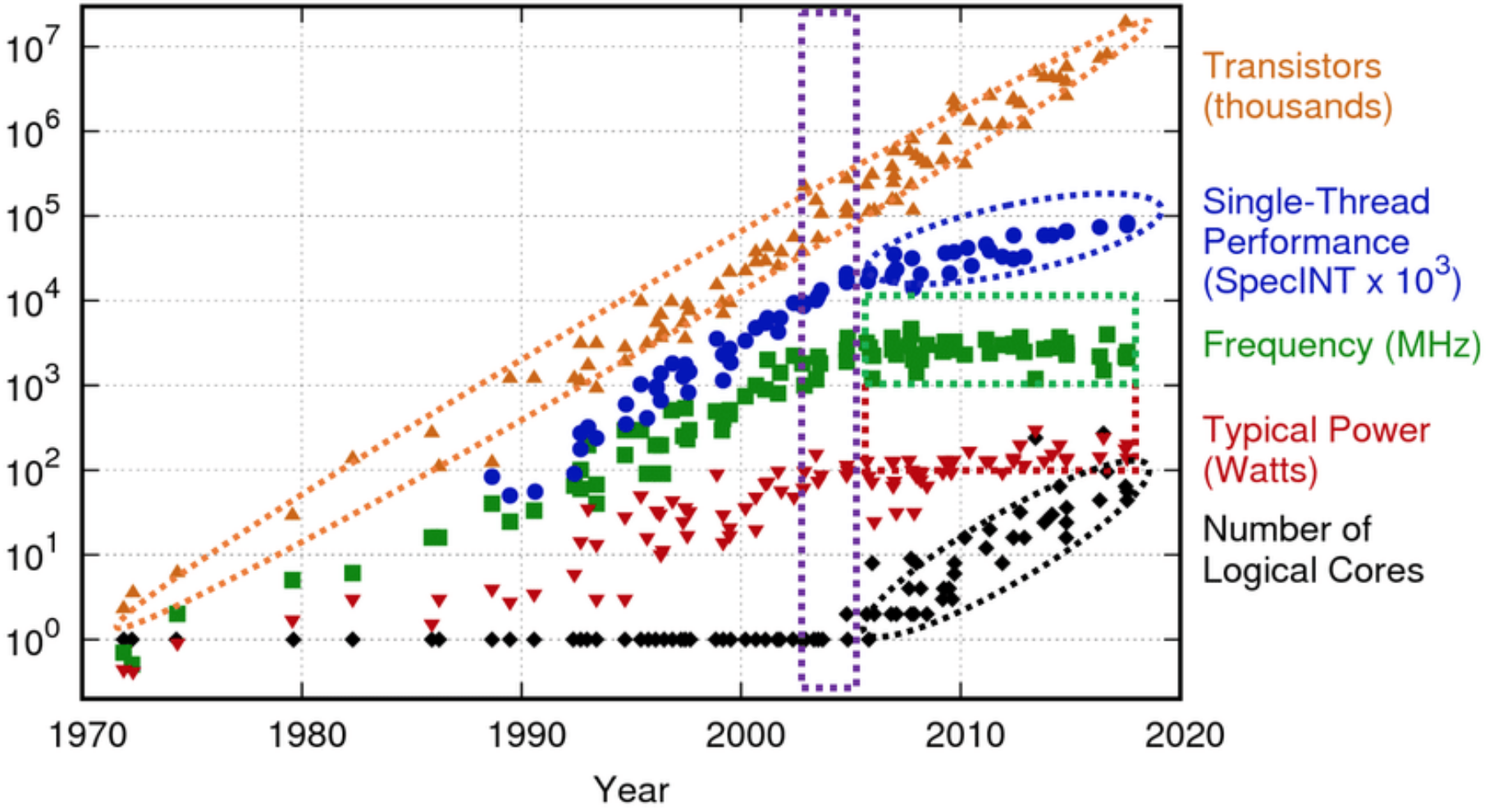
Source: Vishwani Agrawal

The Era of Parallel Computing is here

- Nowadays, performance comes from the software
- There are no smart-enough tools to efficiently parallelize serial software on the parallel hardware
- Free lunch is over…

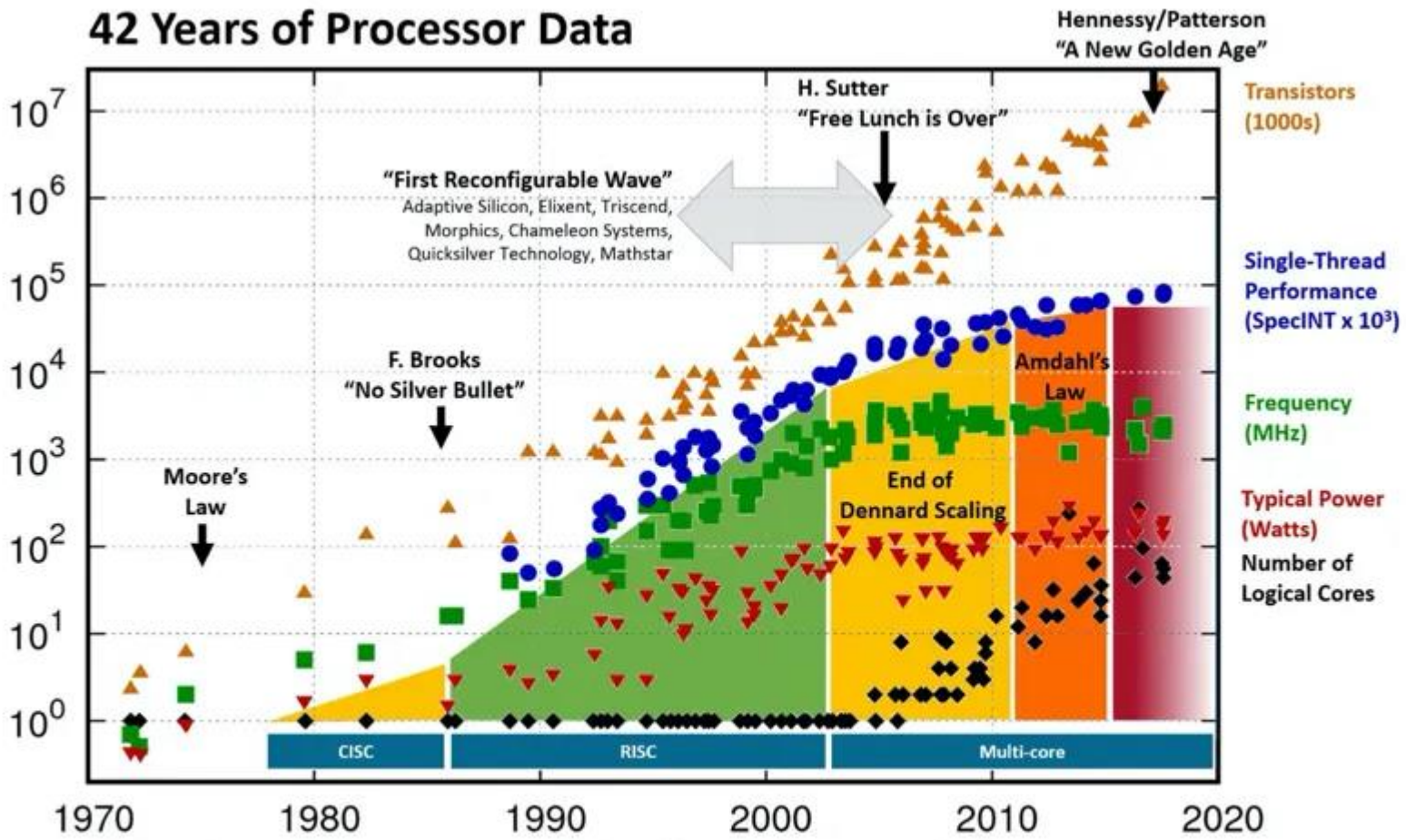
- We must learn how to write parallel applications…

Hardware Architecture Trends (1)



Taken from https://www.researchgate.net/publication/336577121_BACKUS_Comprehensive_High-Performance_Research_Software_Engineering_Approach_for_Simulations_in_Supercomputing_Systems

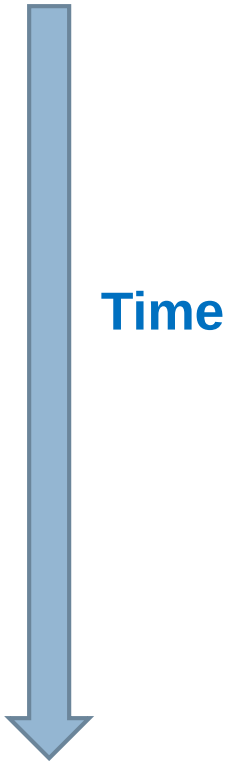
Hardware Architecture Trends (2)



Hennessy and Patterson, Turing Lecture 2018, overlaid over "42 Years of Processors Data"
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/;"First Wave" added by Les Wilson, Frank Schirrmeister
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Hardware Evolution

- Scalar Processors
- Pipelined Processors
- Superscalar and VLIW Processors
- Out of order Processors
- Vectorization
- Hyper-Threading
- Multicore Processors
- Manycore Processors
- Heterogeneous systems



Heterogeneous computing (1)

Single core Era -> Multi-core Era -> Heterogeneous Systems Era

- **Heterogeneous computing refers to systems that use more than one kind of processors or cores**
  - These systems gain performance or energy efficiency not just by adding the same type of processors, but by adding dissimilar (co)-processors, usually incorporating specialized processing capabilities to handle particular tasks
  - Systems with General Purpose Processors (GPPs), GPUs, DSPs, ASIPs etc.

- **Heterogeneous systems offer the opportunity to significantly increase system performance and reduce system power consumption**

Heterogeneous computing (2)

- Software issues:
  - Offloading
  - Programmability – think about CPU code (C code), GPU code (CUDA), FPGA code (VHDL)
  - Portability - What happens if your code runs on a machine with an FPGA instead of a GPU

Comparisons between Homogeneous and Heterogeneous Computing

Symmetric, Same cores (Usually CPUs)	Assymmetric, Different cores (CPUs, GPUs, DSPs and accelerators)
operation is guaranteed to be same at each core	operation cannot be supposed to be same at each core
easy to off load tasks	more complicated to off load tasks
good compatibility	less compatibility **specialized for specific tasks**

Hardware Trends
From single core processors to heterogeneous systems on a chip

Driven by
frequency scaling
Constrained by
power consumption

Single-Core

1975-2005

Multicore

since 2005

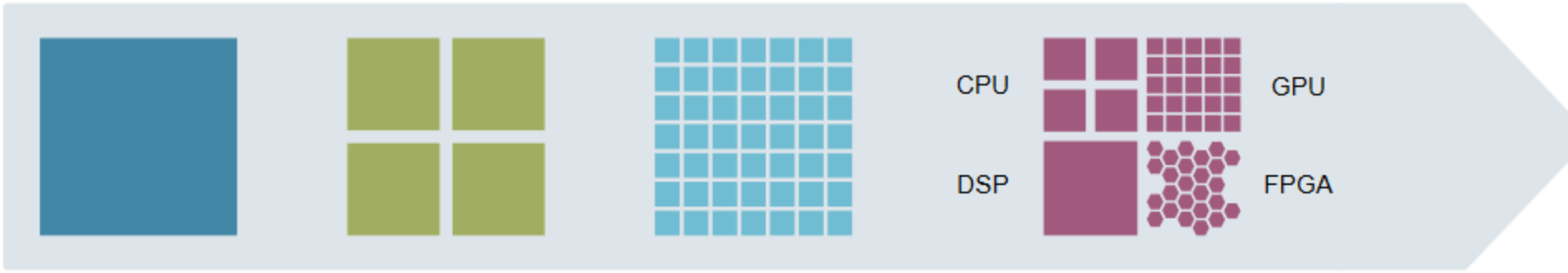
Manycore

since 2008

Heterogeneous systems

today

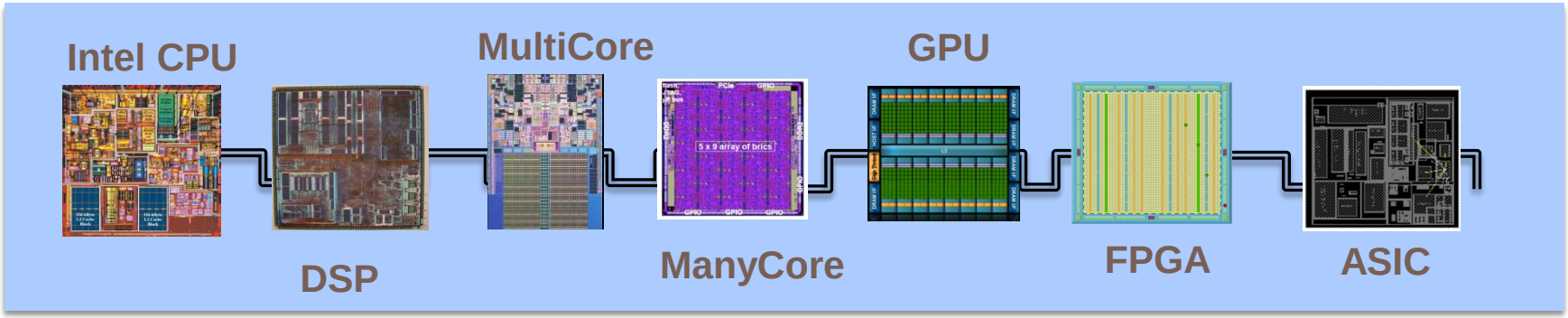
Driven by
high performance per Watt ratio
Constrained by
programming complexity



H. Esmaeilzadeh et al., "Dark silicon and the end of multicore scaling", International Symposium on Computer Architecture (ISCA). ACM, 2011.
M. Zahran, "Heterogeneous Computing Here to Stay". ACM Queue, Nov/Dev 2016.

Taken from https://embb.io/downloads/MTAPI_EMBB.pdf

Comparison of Hardware Architectures



Flexibility, Programming Abstraction

Performance, Area and Power Efficiency

CPU:
- Market-agnostic
- Accessible to many programmers (Python, C++)
- Flexible, portable

FPGA:
- Somewhat Restricted Market
- Harder to Program (VHDL, Verilog)
- More efficient than SW
- More expensive than ASIC

ASIC
- Market-specific
- Fewer programmers
- Rigid, less programmable
- Hard to build (physical)

High Performance Computing (HPC) Programming Languages

- HPC is all about performance
- The most used languages in HPC are
  - C/C++
  - Fortran – there are many old massive applications which are still running, e.g., weather forecast (MetOffice)
- In this module we will be using C Language

Parallel Programming Models/Frameworks/Libraries

- There are too many parallel programming models to write parallel applications
  - Which one to use?
    - Ease of use
    - Performance
    - Portability
- In this module we will be using shared memory architectures only
  - OpenMP
  - CUDA

Serial VS Parallel version
See how elegant OpenMP is


```
double un_opt(){
 int i;
 double x, pi, sum=0.0;
 double step;

  step=1.0/(double) num_steps;

 for (i=0; i<num_steps; i++){
  x=(i+0.5)*step;
  sum = sum + 4.0 / (1.0 + x*x);
}
pi = step * sum;

return pi;
}
```

```
double version6(){
int i;
double x, pi, sum=0.0;
double step;

  step=1.0/(double) num_steps;

#pragma omp parallel for private(x) reduction(+:sum)
for (i=0; i<num_steps; i++){
  x=(i+0.5)*step;
  sum = sum + 4.0 / (1.0 + x*x);
}
pi = step * sum;

return pi;

}
```

GPU Parallel Programming Frameworks

- The main GPU parallel programming frameworks are:
  - **CUDA (Compute Unified Device Architecture)**
    - Only for Nvidia GPUs - Nvidia Corporation proprietary
    - By far Best performance for Nvidia GPUs
  - **OpenCL (Open Computing Language)**
    - Open, maintaned by the Khronos Group
    - Programming is not than different from CUDA
    - Portable - CPUs, GPUs, other coprocessors
  - **OpenMP (Open Multi-Processing) – code annotation**
    - Very easy to use
    - Portable - CPUs, GPUs, other coprocessors
  - **OpenACC (open accelerators) – code annotation**
    - Very easy to use
    - Portable – CPUs, GPUs, other coprocessors

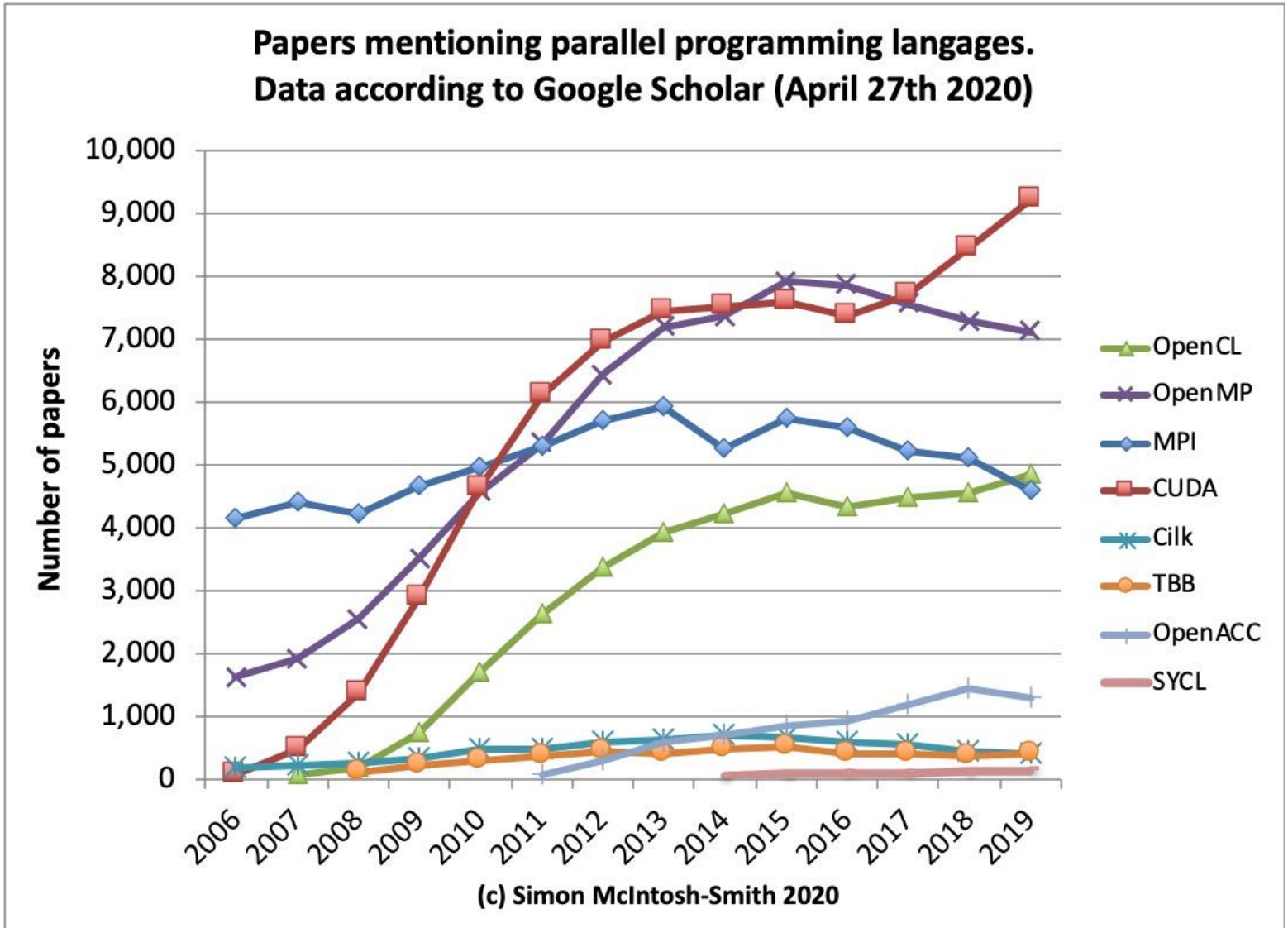
Vector Addition Example using OpenMP and OpenACC

- ***See how easy it is to write GPU code using OpenMP or OpenACC*** …
  - **But not that fast as CUDA for Nvidia GPUs …**
  - Why?
    - CUDA is designed just for Nvidia GPUs, CUDA code is at a lower level, better control of the hardware resources, allows for code optimizations…

```
// OpenACC code that runs on the GPU
#pragma acc kernels copyout(c[0:n]) copyin(a[0:n], b[0:n])
 for (i=0; i<n; i++) {
   c[i] = a[i] + b[i];
 }

 // OpenMP code that run on the GPU
 #pragma omp target map(to: a[0:N], b[:N]) map(from: c[0:N])
 #pragma omp parallel for
 for (int i = 0; i < N; i++) {
       c[i] = a[i] + b[i];
     }
```

Parallel Programming Languages Popularity in Research (not industry)



New ExaScale hardware architectures have been announced

- ***Exascale computing*** is expected to revolutionize computational science and engineering by providing ***1000x the capabilities*** of currently available computing systems, while having a ***similar power footprin***t.
- The new exascale hardware architectures are heterogeneous
  - CPUs+GPUs (Aurora)
  - CPUs+FPGAs (Arm EPI)
- Although, the exascale supercomputers are currently being developed, with 'Aurora' being the first to be announced by the end of 2021, only a few HPC applications are so far able to fully exploit the capabilities of the current **petascale** systems, mainly because of their limited **scalability.**
- Therefore, efforts for preparing HPC applications for Exascale are needed
  - People with such expertise get highly payed jobs

Profiling Software Applications

- *Types of profiling*
  - **Instrumentation-based profiling**
    - Adds instructions to the target program to collect required information
    - Disadvantage: high overhead that distorts elapsed time
  - **Statistical Profiling - Sampling-based**
    - Profilers periodically interrupt the program's execution to collect information
    - Advantage: very low profiling overhead
    - Disadvantage: low statistical accuracy and possible timing anomalies
  - **Hardware counter (event-based) profiling**
    - Uses special CPU registers to count CPU events
    - Advantage: more detailed information with no profiling overhead

Profiling Software Applications

- In this module we will use the following profilers
  - Gprof
  - Valgrind
  - Perf
  - Intel Vtune – Intel provides a free version of Vtune (Intel's Profiler)
    - GUI is supported

Gprof Profiler

- Gprof profiler is a tool which collects statistics on **serial** programs.
- It works by inserting appropriate code in the beginning and in the end of each function so as to collect information about the execution time.
- You can use in Linux only – Specific instructions in the tutorial
- **When to use:**
  - When you want to find the computationally intensive functions of a serial program

File Edit View Search Terminal Help

```
user01@wave:~/Desktop/comp3001/my/Labs/profiling$ gprof p -b
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 49.49      3.50     3.50        1     3.50     3.50  MMM_default
 49.49      7.01     3.50        1     3.50     3.50  MMM
  0.57      7.05     0.04        1     0.04     3.54  writedata_MMM
  0.14      7.06     0.01        1     0.01     0.01  MVM_default
  0.14      7.07     0.01        1     0.01     0.01  vec_add
  0.14      7.08     0.01        1     0.01     0.01  vec_add_default
  0.14      7.09     0.01        1     0.01     0.02  writedata_vec_add
  0.00      7.09     0.00        1     0.00     0.00  MVM
  0.00      7.09     0.00        1     0.00     0.00  initialize
  0.00      7.09     0.00        1     0.00     0.01  writedata_MVM


                     Call graph


granularity: each sample hit covers 2 byte(s) for 0.14% of 7.09 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    7.09                 main [1]
                0.04    3.50       1/1           writedata_MMM [2]
                3.50    0.00       1/1           MMM [4]
                0.01    0.01       1/1           writedata_vec_add [5]
                0.01    0.00       1/1           vec_add [7]
                0.00    0.01       1/1           writedata_MVM [9]
                0.00    0.00       1/1           initialize [11]
                0.00    0.00       1/1           MVM [10]
-----------------------------------------------
                0.04    3.50       1/1           main [1]
[2]     50.0    0.04    3.50       1         writedata_MMM [2]
                3.50    0.00       1/1           MMM_default [3]
-----------------------------------------------
                3.50    0.00       1/1           writedata_MMM [2]
[3]     49.4    3.50    0.00       1         MMM_default [3]
-----------------------------------------------
                3.50    0.00       1/1           main [1]
[4]     49.4    3.50    0.00       1         MMM [4]
-----------------------------------------------
                0.01    0.01       1/1           main [1]
[5]      0.3    0.01    0.01       1         writedata_vec_add [5]
                0.01    0.00       1/1           vec_add_default [8]
-----------------------------------------------
                0.01    0.00       1/1           writedata_MVM [9]
[6]      0.1    0.01    0.00       1         MVM_default [6]
-----------------------------------------------
                0.01    0.00       1/1           main [1]
[7]      0.1    0.01    0.00       1         vec_add [7]
```

A detailed explanation is provided in the tutorial

Valgrind Tool

- The Valgrind tool suite provides a number of debugging and profiling tools that help you make your programs faster and more correct.
- In this module will be using *Cachegrind tool* of Valgrind.
- Cachegrind simulates how your program interacts with cache hierarchy.
- It **simulates** a machine with independent first-level instruction (I1) and data caches (D1), backed by a unified second-level cache (L2).
- For the CPUs with more than 2 levels of cache Cachegrind simulates the first-level and last-level caches only.
- Therefore, Cachegrind always refers to the Instruction L1, data L1 and Last Level cache.
- **When to use:**
  - When you want to optimize memory accesses

Valgrind Tool – Example Output


```
==11586== I   refs:      1,041,351,336
==11586== I1  misses:            1,108
==11586== LLi misses:            1,101
==11586== I1  miss rate:          0.00%
==11586== LLi miss rate:          0.00%
==11586==
==11586== D   refs:        387,398,347  (244,246,437 rd   + 143,151,910 wr)
==11586== D1  misses:          411,589  (    160,745 rd   +     250,844 wr)
==11586== LLd misses:          405,386  (    154,589 rd   +     250,797 wr)
==11586== D1  miss rate:           0.1% (        0.1%     +         0.2%  )
==11586== LLd miss rate:           0.1% (        0.1%     +         0.2%  )
==11586==
==11586== LL refs:             412,697  (    161,853 rd   +     250,844 wr)
==11586== LL misses:           406,487  (    155,690 rd   +     250,797 wr)
==11586== LL miss rate:            0.0% (        0.0%     +         0.2%  )
```

Perf Tool

- The perf tool offers a rich set of commands to collect and analyze performance and trace data.
  - Supports Hardware counters
- When to use
  - When you want to leverage the hardware counters

Intel Vtune

- Perhaps the most powerful and the easiest to use
  - A GUI is supported
- For Intel Processors only
- When to use
  - If you have an Intel Processor

Choose your next analysis type

Select a highlighted recommendation based on your performance snapshot.

ALGORITHM
- Hotspots
- Anomaly Detection (preview)
- Memory Consumption

MICROARCHITECTURE
- Microarchitecture Exploration 41.7%
- Memory Access

PARALLELISM
- Threading 30.7%
- HPC Performance Characterization

I/O
- Input and Output

ACCELERATORS
- GPU Offload
- GPU Compute/Media Hotspots (preview)
- CPU/FPGA Interaction

PLATFORM ANALYSES
- System Overview
- GPU Rendering (preview)
- CPU/GPU Concurrency (deprecated)
- Platform Profiler

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line:	/home/wave/Desktop/p
Operating System:	5.11.0-34-generic DISTRIB_ID=Ubuntu DISTRIB_RELEASE=20.04 DISTRIB_CODENAME=focal DISTRIB_DESCRIPTION="Ubuntu 20.04.3 LTS"
Computer Name:	wave-ThinkCentre-M910s
Result Size:	3.3 MB
Collection start time:	08:58:40 22/09/2021 UTC
Collection stop time:	08:58:43 22/09/2021 UTC
Collector Type:	Event-based counting driver

Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples to speed up post-processing.

CPU

Name:	Intel(R) Processor code named Kabylake
Frequency:	3.4 GHz
Logical CPU Count:	4

Elapsed Time: 2.170s

IPC:	1.636
SP GFLOPS:	1.669
DP GFLOPS:	0.000
x87 GFLOPS:	0.019
Average CPU Frequency:	3.7 GHz

Logical Core Utilization: 30.7% (1.227 out of 4)

Physical Core Utilization: 30.7% (1.227 out of 4)

Microarchitecture Usage: 41.7% of Pipeline Slots

Retiring:	41.7%	of Pipeline Slots
Front-End Bound:	23.5%	of Pipeline Slots
Back-End Bound:	**28.8%**	**of Pipeline Slots**
Memory Bound:	7.5%	of Pipeline Slots
Core Bound:	21.3%	of Pipeline Slots
Bad Speculation:	6.0%	of Pipeline Slots

Memory Bound: 7.5% of Pipeline Slots

L1 Bound:	3.7%	of Clockticks
L2 Bound:	0.7%	of Clockticks
L3 Bound:	2.8%	of Clockticks
DRAM Bound:	**6.9%**	**of Clockticks**
Store Bound:	0.1%	of Clockticks

Vectorization: 94.2% of Packed FP Operations

Instruction Mix:		
SP FLOPs:	5.6%	of uOps
Packed:	98.5%	from SP FP
128-bit:	98.5%	from SP FP
256-bit:	0.0%	from SP FP
Scalar:	1.5%	from SP FP
DP FLOPs:	0.0%	of uOps
Packed:	2.0%	from DP FP
Scalar:	98.0%	from DP FP
x87 FLOPs:	0.3%	of uOps
Non-FP:	94.2%	of uOps
FP Arith/Mem Rd Instr. Ratio:	0.239	
FP Arith/Mem Wr Instr. Ratio:	0.527	

How to measure the performance of our parallel software?

- Measuring elapsed execution time of a program
  - Real Time (Wall time)
  - User Time – does not include the time spent in OS calls or other processes
  - CPU system time – time spent executing system calls
- Floating Point Operations Per Second (FLOPS)
  - Highly used in HPC
- Speedup

Using Accurate Timers to Measure Execution Time Linux (1)

- **In Linux**: The *clock_gettime* system call allows us to measure the execution time of programs.
- This function uses the following struct.

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

- The *clock_gettime* system call is used as follows:
  - *#include <time.h>*
  - *int clock_gettime(clockid_t clk_id, struct timespec *tp);*
- The second parameter is the time structure.
- The first parameter, clock ID, allows you to specify the clock you are interested in using

Using Accurate Timers to Measure Execution Time Linux (2)

CLOCK_MONOTONIC represents the absolute elapsed wall-clock time

```
#define BILLION 1000000000
struct timespec start, end;

/* measure monotonic time */
clock_gettime(CLOCK_MONOTONIC, &start);/* mark start time */

do_something();

clock_gettime(CLOCK_MONOTONIC, &end);/* mark the end time */

diff = BILLION * (end.tv_sec - start.tv_sec) + end.tv_nsec - start.tv_nsec;
printf("elapsed time = %llu nanoseconds\n", (long long unsigned int) diff);
```

Using Accurate Timers to Measure Execution Time Linux (3)

The ***CLOCK_PROCESS_CPU_TIME_ID*** clock measures *only* the CPU time consumed by the process. If the kernel puts the process to sleep, the time it spends waiting is not counted.

```
/* now the measure CPU time for this process only */
/* the time spent sleeping will not count (but there is a bit of overhead */
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);/* mark start time */

do_something();

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);/* mark the end time */

diff = BILLION * (end.tv_sec - start.tv_sec) + end.tv_nsec - start.tv_nsec;
printf("elapsed process CPU time = %llu nanoseconds\n", (long long unsigned int) diff);
```

Accurate Timers in Visual Studio

- The most accurate timer is the '*high_resolution_clock*' which is supported in C++ only.
  - Keep in mind that you can write C code inside a C++ file

```
#include <chrono>
int main() {
auto start = std::chrono::high_resolution_clock::now();
    Do_something();
auto finish = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> elapsed = finish - start;
std::cout << "Elapsed time: " << elapsed.count() << " s\n";
…
}
```

Using OpenMP timer that works on all the operating systems

- In weeks 5-7 we will be using OpenMP
- OpenMP supports *omp_get_wtime() timer*
  - In Linux you must compile using '-fopenmp' option.

```
#include <omp.h>
int main(){
double start, end;

start=omp_get_wtime();
        routine();
end=omp_get_wtime();

printf("Elapsed Time in seconds is %f",end-start);
…
}
```

How to get an accurate execution time value

- When you run your program in an operating system, other processes run too
- **For single thread programs**: To get an accurate execution time value, make sure the execution time of your code is at least a **few seconds**
- **For multi-threaded programs**: To get an accurate execution time value, make sure the execution time of your code is about 1 minute

- If the execution time of your program is much higher than the sum of the all the others, then the execution time value is considered accurate

FLOPS

- The performance capabilities of supercomputers are expressed using a standard rate for indicating the number of floating-point **arithmetic** calculations systems can perform on a per-second basis.
- **How many FLOPS does the following program achieve if it takes 1sec to execute** (the arrays are of type float) ?
  - There are '*2*1024*1024*' FP arithmetic operations (1 addition and 1 multiplication)
  - FLOPS=2*1024*1024 / 1sec = ***2 Mega FLOPS***

```
for (i=0; i<1024; i++)
  for (j=0; j<1024; j++)
    y[i]+=a[N*i+j] * x[j];
```

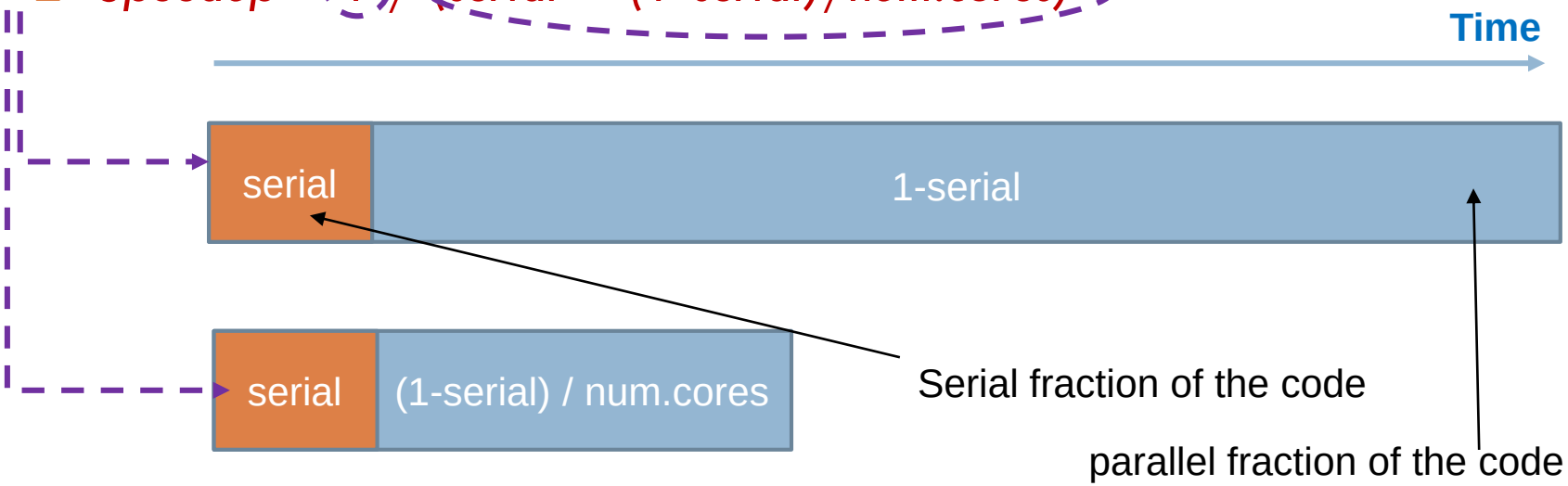
Speedup and Efficiency

- *Speedup = Tserial / Tparallel*
  - The speedup is ideal if *Speedup == num.cores*
  - The speedup is linear if *Speedup ≈ num.cores*
  - In practice it is hard to get linear speedup because of the overhead in creating the threads, communication, resource contention, and synchronization

- *Efficiency = Speedup / num.cores* or *ideal.time/measured.time*
  - It estimates how well-utilized the processors are in solving the problem
  - Ideal speedup gives *Efficiency=1*

Amdahl's Law

- Amdahl pointed out that the speedup is limited by the fraction of the serial part of the application that is not amenable to parallelization
- *Speedup = 1 / (serial + parallel/num.cores)*
  - *Serial/parallel* is the proportion of execution time spent on the serial and parallel part, respectively
  - *Parallel=1-serial* thus:
- *Speedup = 1 / (serial + (1-serial)/num.cores)*



Scalability (1)

- Most parallel applications can be run using different number of processing elements, but the speedup will be decreasing as the number of processing elements increases. For example:
  - 2 cores speedup x1.98
  - 4 cores speedup x3.7
  - 16 cores speedup x9
  - This is normal, but you must carefully choose the number of cores to use
  - Typically, larger input sizes will run efficiently
- ***Scalability or scaling*** is widely used to indicate the ability of hardware and software to deliver greater computational power when the amount of resources is increased

Scalability (2)

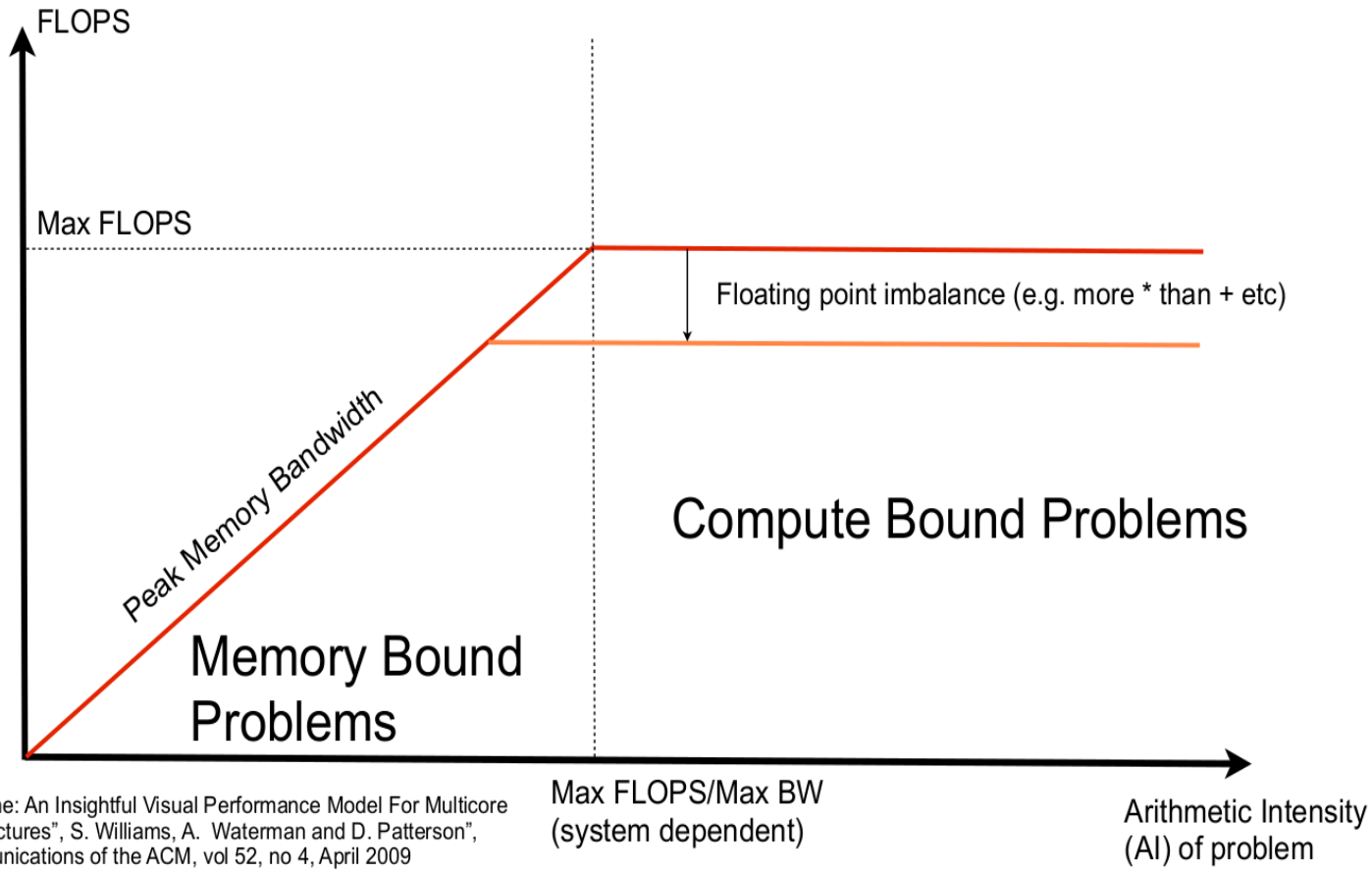
- ***Strong scaling*** refers to an application's performance **when the total problem size is kept fixed,** and the number of **processing elements varied.**
  - What is the speedup if we double the number of cores?
  - What is the speedup if we have x8 cores?
- ***Weak scaling*** refers to an application's performance when we **increase the problem size relative to the number of processing elements.**
  - If we run a program for N=128 on 4 cores, will the time remain constant if we run the program for N=256 on 8 cores?
  - Will the time remain constant since the work per core has remained the same, or will it increase because of the communication overhead / cache misses etc?

The Roofline Model (1)

- **The roofline model provides an easy way to get performance bounds for *compute-bound* and *memory-bound* loop kernels.**
- It allows us to know how far the achieved performance is from the optimum.
- It is based on the concept of computational intensity, sometimes also called arithmetic or operational Intensity.
- The arithmetic intensity (AI) is given by the following formula:
  AI=FP.arithmetical.instructions / number.of.bytes.loaded.stored
- This model has several limitations, e.g., does not consider all features of modern processors and ignores integer computations.
- You can read more in https://people.eecs.berkeley.edu/~kubitron/cs252/handouts/papers/RooflineVyNoYellow.pdf

The Roofline Model (2)

- Algorithms that have a ***low arithmetical intensity are memory-bound***, while algorithms that have a ***high arithmetical intensity are compute-bound***.
- Memory-bound means that their performance is bounded on the memory latency and bandwidth values

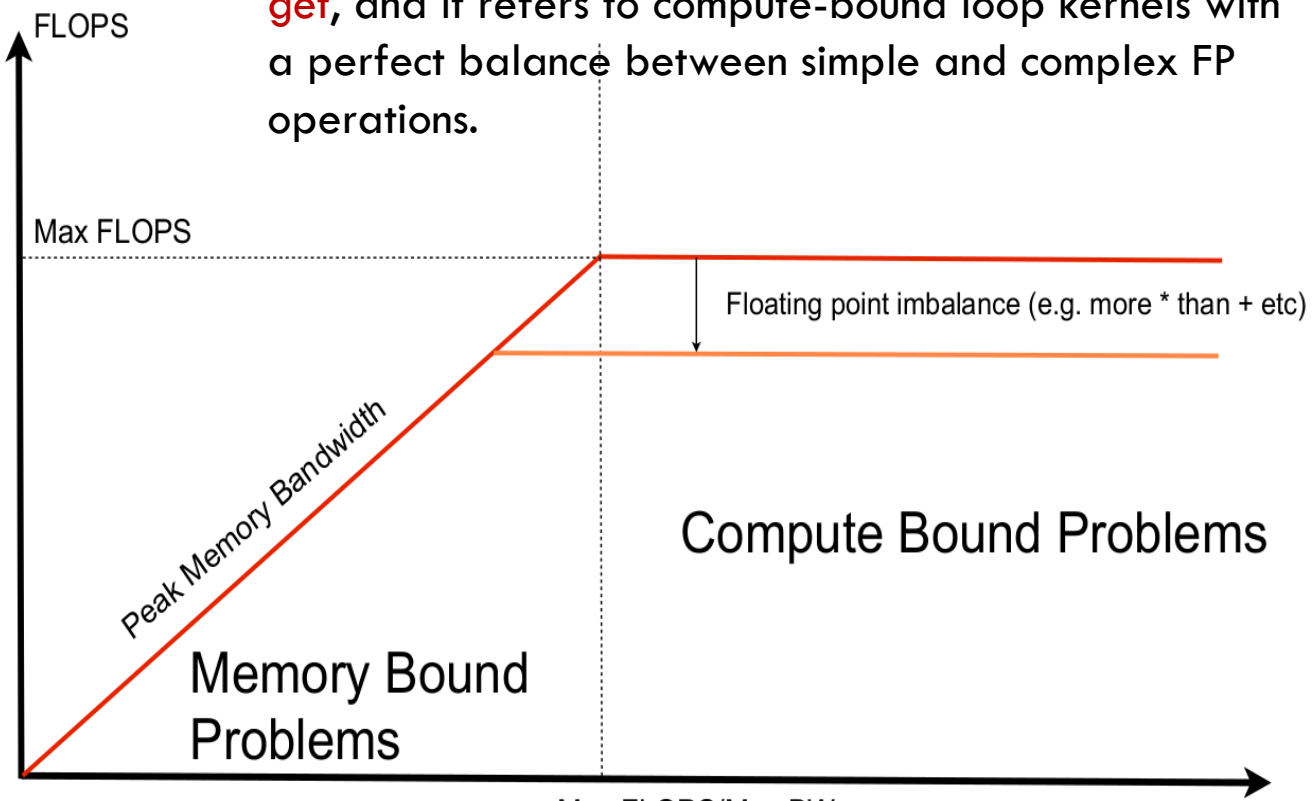


"Roofline: An Insightful Visual Performance Model For Multicore Architectures", S. Williams, A. Waterman and D. Patterson", Communications of the ACM, vol 52, no 4, April 2009

The Roofline Model (3)

FLOPS = min (Peak Floating Point Performance, Peak Memory Bandwidth x AI)

The peak FP performance is the maximum we can get, and it refers to compute-bound loop kernels with a perfect balance between simple and complex FP operations.

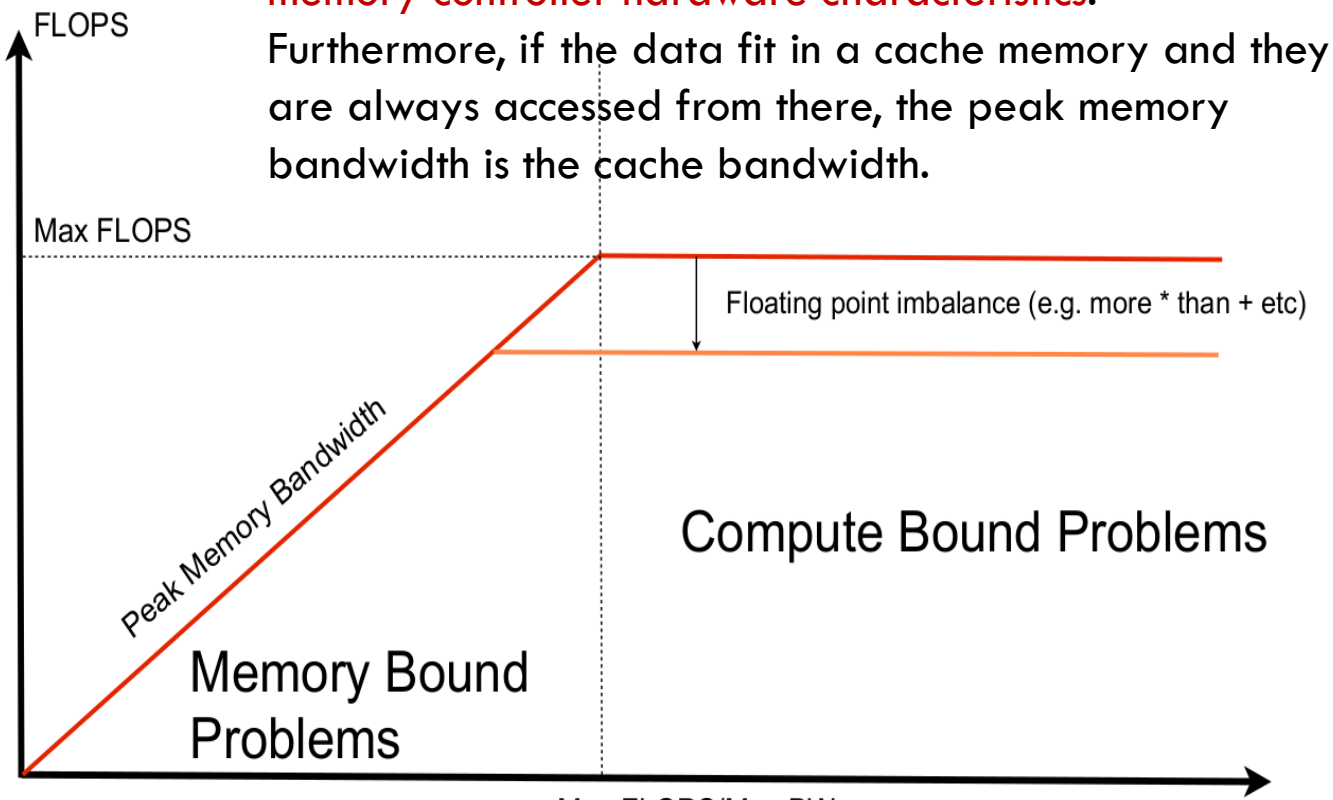


"Roofline: An Insightful Visual Performance Model For Multicore Architectures", S. Williams, A. Waterman and D. Patterson", Communications of the ACM, vol 52, no 4, April 2009

The Roofline Model (4)

FLOPS = min (Peak Floating Point Performance, Peak Memory Bandwidth x AI)

The peak memory bandwidth depends on the DDR and memory controller hardware characteristics.
Furthermore, if the data fit in a cache memory and they are always accessed from there, the peak memory bandwidth is the cache bandwidth.



"Roofline: An Insightful Visual Performance Model For Multicore Architectures", S. Williams, A. Waterman and D. Patterson", Communications of the ACM, vol 52, no 4, April 2009

Roofline Model - Example

- MMM has N^3 iterations and each iteration contains 4 Floating Point (FP) L/S operations and 2 FP arithmetical operations.
- So, the arithmetical intensity of MMM, is 2/(4*4bytes)=1/8.
- Assume that performance is not affected by the integer operations.
- So, if the peak memory bandwidth is 21GBytes/sec, then the maximum MMM performance will be 21GB/sec x (1/8)=2.65gigaflops.
- If the arrays fit in the precious cache memories (using optimizations), then the memory bandwidth is higher and thus performance is increased.

```
for (i=0;i<N;i++)
 for (j=0;j<N;j++)
for (k=0;k<N;k++)
C[i][j] +=A [i][k] * B[k][j];
```

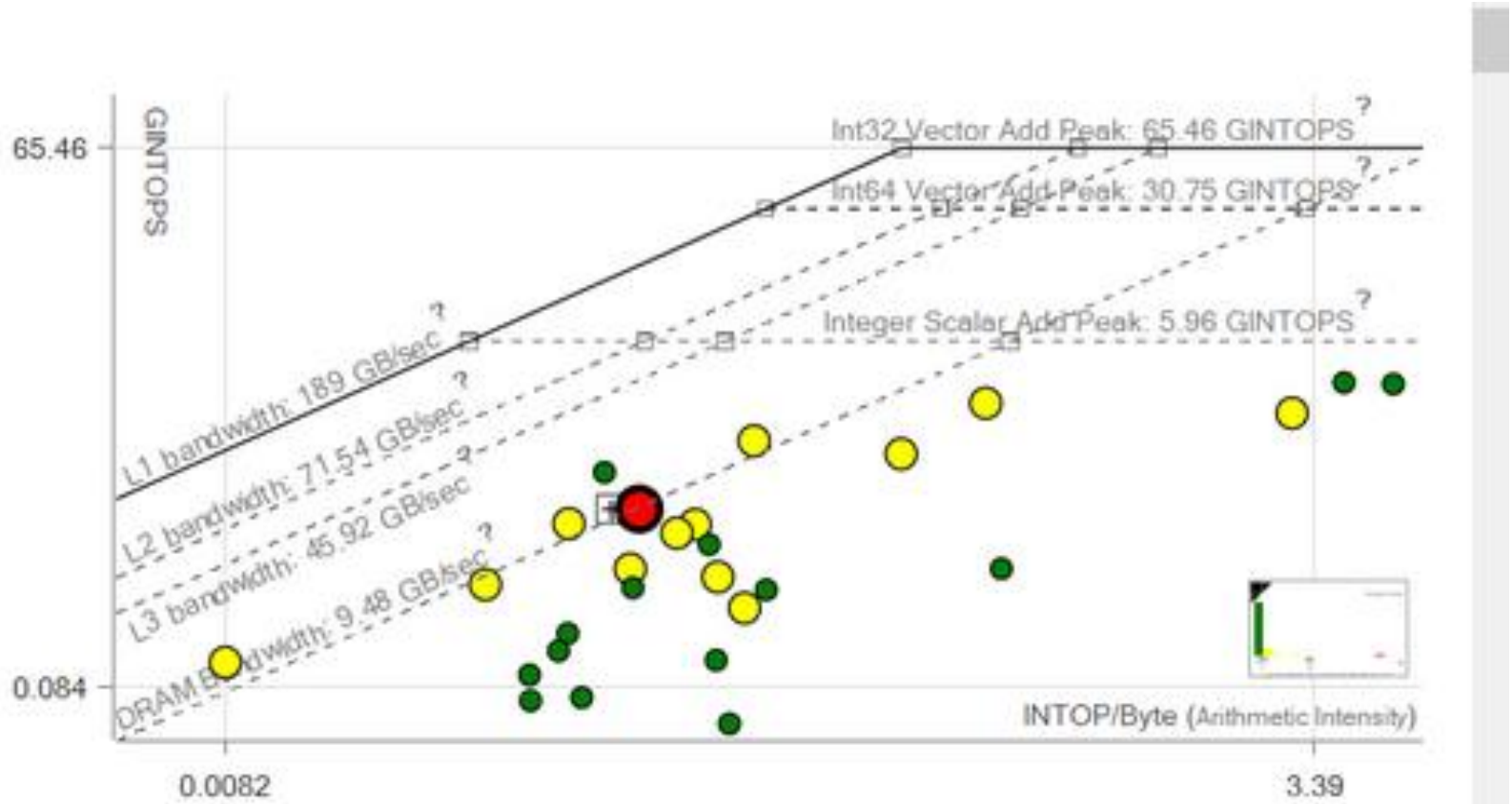
Roofline Model – Example (2)

- Find the Arithmetic Intensity of the following program

```
Float Y[N], A[N][N], X[N];

for (i=0; i<N; i++)
 for (j=0; j<N; j++)
   y[i] +=A[i][j] * X[j];
```

Roofline Model – Example (3)



Intel Advisor. Taken from
https://software.intel.com/content/www/us/en/develop/articles/a-brief-overview-of-integer-roofline-modeling-in-intel-advisor.html

Performance Guidelines

- **Compute bound** codes are not that hard to optimized
  - The speedup achieved will be small
- Speeding up **memory bound** codes is really challenging and researchers still working on it
  - **The main software strategies** are as follows.
    - Reducing the number of memory accesses through the whole memory hierarchy.
    - Use software prefetching.
      - The above can be achieved by using code optimizations such as loop tiling, register blocking, array copying, loop merge/distribution etc. We will study those next week.
- There are **I/O bound problems** too, but not studied here

Further Reading

- GPROF Tutorial – How to use Linux GNU GCC Profiling Tool, available at https://www.thegeekstuff.com/2012/08/gprof-tutorial/
- The Valgrind Quick Start Guide, available at https://www.valgrind.org/docs/manual/quick-start.html#quick-start.intro
- Cachegrind: a cache and branch-prediction, available at https://valgrind.org/docs/manual/cg-manual.html
- Tutorial, Linux kernel profiling with perf, available at https://perf.wiki.kernel.org/index.php/Tutorial
- perf Examples, available at http://www.brendangregg.com/perf.html
- Get started with Intel Vtune, available at https://software.intel.com/content/www/us/en/develop/documentation/get-started-with-vtune/top/windows-os.html