Parallel Processors : Computer architects attempt to create powerful computers by connecting many existing smaller ones. This led to the concept of multiprocessors. A multiprocessor is a computer system with at least two processors. Ideally, customers order as many processors as they can afford, and like to receive a proportional amount of performance. Thus, multiprocessor software must be designed to ~~work~~ work with a variable number of processors. Recall that energy has become very important issue for both microprocessors and data centres (a datacentre is a large group of networked computer servers typically used by organizations for the remote storage, processing or distribution of large amounts of data). Replacing large inefficient processors with many smaller, efficient processors can deliver better performance per Joule both in the large (computer) as well in the small (computer), if software can efficiently use them. Thus, improved energy efficiency rather multiprocessor joins scalable performance in the case for multiprocessors.

- Since multiprocessor software should scale, some designs support operation in the presence of broken hardware. Thus, if a single processor fails in a multiprocessor with n processors, this system should continue to provide service with n-1 processors. Hence, multiprocessors can also improve availability.
- High performance can mean high throughput for independent tasks, called task-level parallelism or process-level parallelism. These tasks are independent single-threaded applications, and they are an important and popular use of multiple processors. This approach is in contrast to running a single job on multiple processors. One uses the term parallel processing program to refer to a single program that runs on multiple processors simultaneously.
- There have long been scientific problems that have needed much faster computers, and this class of problems has been used to justify many novel parallel computers over the decades. Some of these problems can be solved today by using a cluster composed of microprocessors stationed in many independent servers. In addition, clusters can serve equally demanding applications outside the sciences, such as search engines, web servers, email servers, and databases.

⇒ **Reason of Switch from Uniprocessor to Multiprocessor**

- Looking at the increase in clock rate and power of nine generations of Intel microprocessors over 36 years (from 1982 to 2018), both clock rate and power increased rapidly for decades (Intel 80286 (1982) had clock rate

P.T.O.

of 12.5 MHz and power 3.3 watts while Intel Core i5 Coffee Lake (2018) has clock rate 3.6 GHz and power 95 watts?, and then flattened or dropped off recently. The reason they grew together is that they are correlated, and the reason for their recent slowing is that we have run into the practical power limit for cooling commodity microprocessors.

[Although power provides a limit to what we can cool, in the PostPC Era, energy is more critical resource than ~~energy~~ power. Battery life can outpace performance in the personal mobile device, and the architects of warehouse scale computers (WSC) try to reduce the costs of powering and cooling 50,000 servers, as the costs are high at this scale.

CMOS is the dominant technology for integrate circuits. For CMOS, the primary source of energy consumption is so-called dynamic energy - that is, energy that is consumed when transistor switch states from 0 to 1 and vice versa. The dynamic energy depends on capacitive loading of each transistor and voltage applied:

Energy $\propto$ capacitive load $\times$ Voltage^2

The above equation is energy of a pulse during the logic transition of $0 \to 1 \to 0$ or $1 \to 0 \to 1$.

Energy of a single transistor is then

Energy $\propto \frac{1}{2} \times$ cap. load $\times$ Voltage^2

Power required per transistor:

Power $\propto \frac{1}{2} *$ cap. load $\times$ Volts$^2 *$ Frequency switched

Frequency of switching is a function of clock rate. The capacitive load per transistor is a function of both the number of transistors connected to an output (called fanout) and the technology, which determines the capacitance of both wires and transistors.

Question: How could clock rates grow by a factor of $\sim 300 \left(\frac{3600}{12.5}\right)$ while power grew by only a factor of $30 \left(\frac{95}{3.3}\right)$? Energy & power can be reduced by lowering the voltage, which occurred with each new generation of technology, and power $\propto$ voltage^2. Typically, voltage was reduced about 15% per generation. In 20 years, voltages have gone from 5V to 1V, which is why the increase in power is only 30 times.

- Problem today is: further lowering of voltage appears to make the transistors too leaky (even today about 40% of power consumption in server chips is due to leakage). To try to address the power problem, designers have already attached large devices to increase cooling, and they turn off parts of the chip that are not used in a given clock cycle.
- Since computer designers have hit a power wall, they needed a new way forward. They chose a different path from the way they designed microprocessors for their first 30 years

Parallel Processors Contd,

The power limit has forced a considerable change in the design of microprocessors. Since 2002, the rate of improvement in response time of programs for desktop microprocessors (over time) has slowed from a factor of 1.5 per year to a factor of only 1.03 per year. Rather than continuing to decrease the response time of a single program running on the single processor, since 2006 all desktop and server companies have been shipping microprocessors with multiple processors per chip, where the benefit is often more on throughput than on response time. To reduce confusion between the words processor and microprocessor, companies refer to processors as "cores", and such microprocessors are generally called multicore microprocessors. Thus, a "quadcore" microprocessor is a chip that contains four processors or four cores.

- In the past, programmers could rely on innovations in hardware, architecture, and compilers to double performance of their programs every 18 months without having to change a line of code. Today, for programmers to get significant improvement in response time, they need to rewrite their programs to take advantage of multiple processors. Moreover, to get the historic benefit of running faster on new microprocessors, programmers will have to continue to improve performance of their code as the number of cores increases
- Thus, future increase in performance (of a computer) must come from some place other than much higher clock rates or improved CPI (cycles per instruction) because of the energy problem.
- A multicore microprocessor is a microprocessor containing multiple processors (cores) in a single integrated circuit. Virtually all microprocessors today in desktops and servers are multicore. The number of cores is expected to increase with improved hardware technology. These multicores are almost always shared Memory Processors (SMPs), as they usually share a single physical address space.
- The state of technology today implies that programmers who care about performance must become parallel programmers. The challenge facing the industry is to create hardware and software that will make it easy to write correct parallel processing programs that will execute efficiently in performance and energy as the number of cores per

P.T.O.

@ chip scales.

• Abrupt shift in microprocessor design brought with it much confusion about the terminology (and what it means!). The figure below tries to clarify the terms serial, parallel, sequential and concurrent. The columns of this figure represent the software, which is either inherently sequential or concurrent. The rows of the figure represent the hardware, which is either serial or parallel.

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in Matlab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in Matlab running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

Figure: Hardware/software categorization and examples of application perspective on concurrency versus hardware perspective on parallelism

For example, the programmers of compilers think of them as sequential programs: the steps include lexical analysis, syntactic analysis (a.k.a parsing), semantic analysis, intermediate code generation, code optimization, and code generation. In contrast, the programmers of operating systems normally think of them as concurrent programs: cooperating processes handling I/O events due to independent jobs running on a computer.

• The point of the two axes of the above figure is that concurrent software can run on serial hardware, such as operating systems for the Intel Pentium 4 uniprocessor, or on parallel hardware, such as an OS on the more recent Intel Core i7. The same is true for sequential software. For example, the MATLAB programmer writes a matrix multiply program thinking about it sequentially, but it can run serially on the Pentium 4 or in parallel on the Intel Core i7.

[Note: in Intel Core i7, there are four independent cores (a.k.a processors) with Hyper-threading (a form of simultaneous multiple threading technology introduced by Intel, though the concept was patented earlier by Sun Microsystems) which supports two heavyweight threads (processes) per processor, presenting the abstraction of two independent logical (virtual) processors), an Intel core i7 multicore µp presents eight processors to any software. An Intel Pentium 4 processor, which is a single-core µp, is capable of running Hyper-threading, and thereby allows a concurrent program to be run on it. Windows Vista OS uses concurrent programming features - Note that concurrent

Parallel Processing (Cont.)

programming is the act of running several programs (apparently) simultaneously, achieved by executing small sections from each program in turn. Concurrent computing is the concurrent (simultaneous) execution of multiple interacting computational tasks. These tasks may be implemented as separate programs, or as a set of processes or threads created by a single program. The tasks may also be executing on a single processor, several processors (in close proximity), or distributed across a network

Apparently, the only challenge of parallel processing is figuring out how to make naturally sequential software have high performance on parallel hardware; but it is also to make concurrent programs have high performance on multiprocessors, as the number of processors increases. One can use (the term) parallel processing program or parallel software to mean either sequential or concurrent software running on parallel hardware

Difficulty of creating Parallel Processing Programs

- the challenge with parallelism is not the hardware; it is that too few important application programs have been rewritten to complete tasks sooner on multiprocessors. It is difficult to write software that uses multiple processors to complete one task faster, and the problem gets worse as the number of processors increases.
- Question: why have parallel processing programs been so much harder to develop than sequential programs? First reason is that one must get better performance or better energy efficiency from a parallel processing program or a multiprocessor; otherwise, one would just use a sequential program on a uniprocessor, as sequential programming is simpler. In fact, uniprocessor design techniques such as superscalar and out-of-order (or dynamic) execution take advantage of instruction-level parallelism, normally without the involvement of the programmer. Such innovations reduced the demand for rewriting programs for multiprocessors, since programmers could do nothing and yet their sequential programs would run faster on new computers.
- Question: why is it difficult to write parallel processing programs that are fast, specially as the number of processors increases?

PTO

Consider eight reporters trying to write a single story in the hope of doing the work eight times faster. To succeed, the task must be broken into eight equal-sized pieces, because otherwise some reporters ~~with~~ would be idle while waiting for the ones with larger pieces to finish. Another obstacle (to speed-up) could be that the reporters would spend too much time communicating with each other instead of writing their pieces of the story. For both this analogy and parallel programming, the challenges include scheduling, partitioning the work into parallel pieces, balancing the load evenly among the workers, time to synchronize, and overhead for communication between the parties. The challenge is stiffer with more reporters for a newspaper story and with more processors for parallel programming.

* Amdahl's law poses another obstacle. It reminds us that even small parts of a program must be parallelized if the program is to make good use of many cores.

Speed-up Challenge

Example: Suppose one wishes to achieve a speed-up of 90 times faster with 100 processors. What percentage of the original computation can be sequential?

Soln. Speedup = $\frac{\text{Execution time (ET) before improvement}}{\text{Execution time (ET) after improvement}}$

Or, Speed-up = $\frac{\text{E.T. before improvement}}{\frac{\text{E.T. affected (after improvement)}}{\text{Amount of improvement}} + \text{E.T. unaffected}}$

$= \frac{\text{E.T. before improvement}}{\frac{\text{E.T. affected}}{\text{Amount of Improvement}} + (\text{E.T. before} - \text{E.T. affected})} \quad \ldots (1)$

Let us rewrite the formula by assuming that E.T. before = 1 (for some unit of time) and ET affected by improvement is considered the fraction of the original E.T.

Thus, dividing numerator and denominator in (1)

Speed-up = $\frac{\text{E.T. before} / \text{E.T. before}}{\frac{\text{E.T before}}{\text{E.T. before}} - \frac{\text{E.T. affected}}{\text{E.T. before}} + \frac{\text{ET affected}}{\text{Amount of improvement} * \text{E.T. before}}}$

Or, Speed-up = $\frac{1}{(1 - \text{Fraction time affected}) + \frac{\text{Fraction time affected}}{\text{Amount of improvement}}} \quad \ldots (2)$

Speed-up challenge (cont)

$$\text{Speed-up} = \frac{1}{(1 - \text{fraction time affected}) + \frac{\text{fraction time affected}}{\text{Amount of improvement}}} \quad \ldots (2)$$

In eqn (2), (formula for speed-up), substitute 90 for speed up and 100 for amount of improvement, to get

$$90 = \frac{1}{(1 - \text{fraction time affected}) + \frac{\text{fraction time affected}}{100}}$$

(multiplying numerator & denominator by 100 and abbreviating fraction time affected as FTA)

$$\text{or, } 90 = \frac{100}{100(1 - FTA) + FTA}$$

$$\text{or, } 9000(1 - FTA) + 90\,FTA = 100$$

$$\text{or, } 8900 = 8910\,FTA \quad \therefore FTA = \frac{8900}{8910} \simeq 0.9989 = 0.999$$

Thus to achieve a speed-up of 90 using 100 processors, the percentage of sequential computation can be $(1 - 0.999) \times \frac{100}{100} = 0.001 \times \frac{100}{100}$

$= 0.1\%$

Speed-up Challenge : Bigger Problem

Suppose one wants to perform two sums: one is a sum of 10 scalar variables, and the other is a matrix sum of a pair of 2-dimensional arrays, with dimensions 10 by 10. For now, let us assume that only the matrix sum is parallelizable. What speed-up does one get with 10 versus 40 processors? Next, calculate the speed-ups assuming that the matrices grow to 20 by 20.

Solⁿ: Assume that performance is a function of time for an addition, which is t. Then there are 10 additions that do not benefit from parallel processors, and 100 additions that do. If the time for a single processor is $110t$, the execution time for 10 processors is

$$\text{Execution time after improvement} = \frac{\text{E.T. affected by improvement}}{\text{Amount of improvement}} + \text{E.T. unaffected}$$

$\therefore$ E.T. after improvement $= \frac{100t}{10} + 10t = 20t$

So, the speed-up with 10 processors is $\frac{110t}{20t} = \boxed{5.5}$

Now, for 40 processors, E.T. after improvement $= \frac{100t}{40} + 10t = 12.5t$

So, the speed-up with 40 processors is $\frac{110t}{12.5t} = \boxed{8.8}$

P.T.O.

- If, potential speedup with 10 processors is 10, then one got $\frac{5.5}{10} \times \frac{100}{100} = 55\%$ of potential speed-up with 10 processors.
- Similarly, if potential speedup with 40 processors is 40, then one gets $\frac{8.8}{40} \times \frac{100}{100} = 22\%$ of the potential speed-up with 40 processors.

- Now, let us increase the matrix to 20 by 20 (which has 400 elements in all). The sequential program now takes total time of $10t + 400t = 410t$

The execution time (E.T.) for 10 processors is

E.T. after improvement $= \frac{400t}{10} + 10t = 50t$

So, the speed-up with 10 processors is $\frac{410t}{50t} = \boxed{8.2}$

Now, execution time for 40 processors is

E.T. after improvement $= \frac{400t}{40} + 10t = 20t.$

So, the speed-up with 40 processors is $\frac{410t}{20t} = \boxed{20.5}$

Thus for this larger problem (of 20×20 matrix) size, one gets

$\frac{8.2}{10} \times \frac{100}{100} = 82\%$ of potential speedup with 10 processors,

while $\frac{20.5}{40} \times \frac{100}{100} = 51.25\%$ of potential speed-up with 40 processors

- The above examples show that getting good speed-up on a multiprocessor while keeping the problem size fixed is harder than getting good speed-up by increasing the size of the problem.

This understanding allows us to introduce two terms that describe ways to scale up.

1) Strong scaling: one measures speed-up which is achieved on a multiprocessor without increasing the size of the problem, i.e. by keeping the problem size fixed.

2) Weak scaling: one measures speed-up which is achieved on a multiprocessor while increasing the size of the problem proportionally to the increase in the number of processors.

speed-up challenge : Balancing Load

• To achieve the speedup of 20.5 on the previous larger problem (working with matrix sum for 20×20 matrix and a sum of 10 scalar variables) with 40 processors, it is assumed that the load was perfectly balanced. That is, each of the 40 processors had ~~400~~ $\frac{100}{40} = 2.5\%$ of the work to do.

Now, assume that one processor's load is higher than all the rest. calculate at twice the load (5%) and five times the load ~~(50%)~~ (12.5%) for that hardest working processor. How well utilized are the rest of the processors?

Solⁿ: Case 1: if one processor has 5% of the parallel load, then it must do 5% × 400 or 20 additions, and the other 39 (processors) will share the remaining 380 (additions). Since they operate simultaneously, one can just ~~calcul~~ calculate the execution time as a maximum

Exec. time after improvement $= \text{Max}\left\{\frac{380t}{39}, \frac{20t}{1}\right\} + 10t = 30t$

Thus, the speed-up drops from 20.5 to $\frac{410t}{30t} \simeq 14$. The remaining 39 processors are utilized less than half the time; while waiting for 20t (units of time) for the hardest working processor to finish, they only compute for $\frac{380t}{39} \simeq 9.7t$

Case 2: if one processor has 12.5% of the load, it must perform

$\frac{12.5}{100} \times 400 = 50$ additions. Then,

E.T. after improvement $= \text{Max}\left\{\frac{350t}{39}, \frac{50t}{1}\right\} + 10t = 60t$

So, speed-up drops even further to $\frac{410t}{60t} \simeq 7$ (from 20.5). The rest of the processors are utilized less than 20% of the time (as $\frac{350t}{39} \simeq 9$, and $\frac{9}{50} \simeq 18\%$)

• This example demonstrates the importance of balancing load, for just a single processor with twice the load of the others cuts speed-up by a third (i.e. from $20.5 \simeq 21$ to 14), and five times the load on ~~just~~ just one processor reduces speed-up by almost a factor of three (i.e. from $20.5 \simeq 21$ to 7)

P.T.O.