Granularity & Complexity in Parallel Systems

Mihaela Maliţa
Smith College, Northampton, MA
mmalita@comcast.net

Gheorghe Ştefan
Politehnica University of Bucharest, Romania
gstefan@gemicer.com

ABSTRACT
Computational performances for parallel architecture could be defined only application domain dependent. Building efficient parallel machines supposes supra-linear computational performances. An efficient parallel system must correlate its granularity with the complexity of its elements. The **supra-linear condition** is introduced to define the conditions for a parallel system to have supra-linear performances. Results show that the basic rule governing large computational systems is that **their complexity must grow slower than their size** because **computation is a simple process**. The distinction between size and (algorithmic) complexity is defined.

KEY WORDS
parallel computation, size, (algorithmic) complexity, granularity.

1 The basic conjecture

Parallel computation has two extreme approaches:

- distributed computing (interconnecting many general purpose computers into a general purpose computing network)
- parallel machines (composed from many computational devices on a chip, board, box, ...)

Usually an under-linear performance is obtained using a network of distributed computers (Figure 1). We do not care too much about the efficiency of distributed computing because we use a multipurpose system to implement it. But, if a machine is specially designed to perform parallel computation it must be efficient in order to make sense to develop specific technologies and to build it. The main question occurs: *how can we design an efficient parallel machine?* A possible answer: *building a supra-linear parallel machine in order to compensate the additional costs involved.*

There are two main approaches for building parallel machines: (1) using many big and complex powerful machines, or (2) connecting together many small and simple powerless machines.

> **Conjecture**: *for an efficient parallel machine when the number of machines increases their size and complexity must decrease.*

In this conjecture two terms must be carefully distinguished:

size : measures the number of gates used to build a digital system *or* the area on silicon to build a digital system

complexity : is proportional with the size of the description used to define a digital system (i.e., according to [1], the size of the program[1] writ-

[1]Gregory Chaitin defined the algorithmic complexity of a binary string as being proportional with the shortest program which generate it.

ten in a Hardware Description Language which describes the digital system, for example)

In **One Billion Transistor Per Chip Era** the confusion between the *size* and the *complexity* of a digital system can not be tolerated anymore.

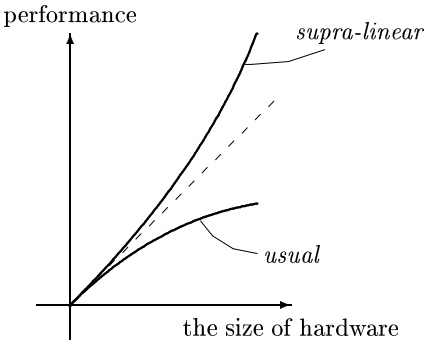

Figure 1: Supra-linear/under-linear performance in parallel computation.

In distributed computing domain the distinction MIMD/SIMD still works, even if most of parallel applications suppose the SIMD approach. But, the same distinction becomes obsolete applied to parallel machines in which the **cellular** approach (a sort of SIMD) fits better.

The cellular approach supposes a fine grain parallel computation. Therefore, the care for the size and complexity of each cell makes sense for defining an optimal machine.

2 Looking for supra-linear performances

Parallelism means more hardware. Putting together more circuits some specific problems occur. The price of the die increases more than proportional with the die's size. The clock frequency depends on the die's area or on the number of dies. The supra-linear performances are needed in order to compensate these effects.

A powerful machine is a big size complex machine which performs *any* operation (simple or complex) in constant time (usually in one clock cycle). Let us call it *big-machine*.

A less powerful machine is a small and simple machine which perform only *some* (simple) operations in one clock cycle, while the complex operations are performed in many clock cycles. Let us call it *small-machine*.

Putting together N big-machines the resulting peak performances will increase at most N times on an area N times bigger. Big-machines can not be used to obtain supra-linear peak performances. Our problem is to find a solution for designing a machine with an area N times bigger but *more than* N times powerful.

2.1 The strategy

A simplified strategy for designing a parallel machine is proposed. The basic assumption of the strategy is: *the performance of a parallel machine is application domain dependent.* The dynamic weight of complex operations depends on the application. For example, in implementing the MPEG compression algorithm the dynamic weight of multiplication operations is less than in implementing the JPEG compression algorithm (the sum of absolute differences computed for the MPEG algorithm does not use multiplications).

Let us consider OPC (Operations Per Cycle) for the two kind of machines:

$$OPC_{big\_machine} = 1$$

$$OPC_{small\_machine} = \frac{p}{p-1+q}$$

where, the application domain is characterized by:

$$p = \frac{total\_number\_of\_operations}{number\_of\_complex\_operations}$$

and the small-machine is characterized by:
q - the number of cycles needed to perform a complex operation on a small-machine.

The performance of a small-machine increases if the application supposes complex operations with a small weight (big value for p), and if efficient algorithms are used to implement the complex operations (small value for q).

OPC for a small-machine Let us consider the following scenario:

- multiplication is the only complex operation for a small-machine. It is performed in $q = 11$ clock cycles.
- for a specific application at each $p = 5$ operations only one operation is a multiplication.

Results: $OPC_{small\_machine} = 0.33$.

◇

The two extreme solutions for building a parallel machine are represented in Figure 2 and Figure 3. The first is a structure containing N big-machines. The second is expanded in the data section of one big-machine as a network of n small-machines. It is supposed that the communication mechanisms in the network of small-machines are very simple (the main lesson learned from [2]) involving almost no circuits. Also, the total area of the n small memories in the second version is similar with the total area of N bigger memories of the first version.

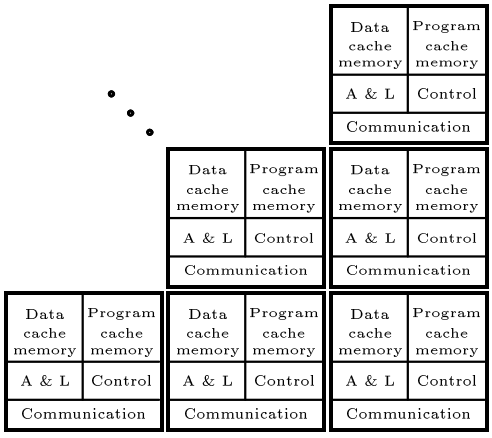

Figure 2: MIMD approach: interconnecting big-machines.

The following notations are used:

A_P : area of a big-machine

$A_{A\&L}$: area of the *arithmetic & logic* resources of a big-machine

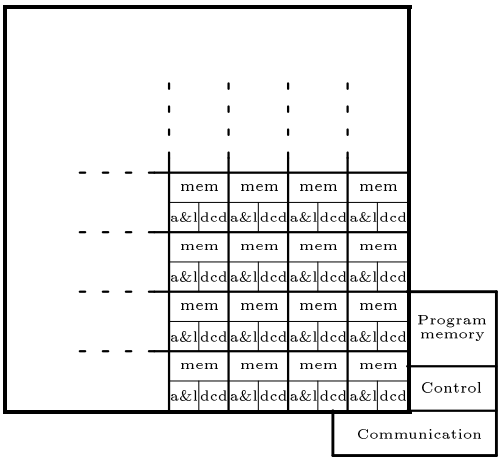

Figure 3: SIMD approach: interconnecting small-machines.

A_p : area of a small-machine

$A_{a\&l\&dcd}$: area of the *arithmetic & logic & decoding* resources of a small-machine

$R = \frac{A_P}{A_{A\&L}}$

$r = \frac{A_{A\&L}}{A_{a\&l\&dcd}}$

According with the previous suppositions, if on the area of N big-machines (Figure 2), equal with

$$N \times A_P$$

we built a network of small-machines (Figure 3), the resulting machine has the performance:

$$OPC_{network\_of\_small\_machines} =$$

$$(1 + (N-1)R) \times r \times OPC_{small\_machine} =$$

$$(1 + (N-1)R) \times r \times \frac{p}{p-1+q}$$

Results n, the total number of small-machines on the area of N big-machines:

$$n(N) = (1 + (N-1)R)r$$

In order to obtain supra-linear performances on an area of N big-machines with a network of n small-machines the following condition must be fulfilled:

$$(1 + (N-1)R)r\frac{p}{p-1+q} > N$$

In order to have positive solutions for the equation:

$$(1 + (N-1)R)r\frac{p}{p-1+q} = N$$

which gives us the minimum number of $n(N)$ for which the supra-linear performances are obtained, the following condition is needed:

$$\frac{A_P}{A_{a\&l\&dcd}} > 1 + \frac{q-1}{p}$$

Let us call it the *supra-linear condition.* We interpret it saying that supra-linear performances suppose:

- big R: the big-machines are big sized and complex
- big r: an efficient design for the small-machines
- small q: efficient algorithms for the complex functions implemented on the small-machines
- big p: a "friendly" application, with a reasonable weight of complex operations.

When the supra-linear condition does not apply Let be the following case:
$R = 1.5;\ r = 3;\ q = 30;\ p = 3.$
In these conditions a super-linear performance is not possible because q is too big (the small-machines are too weak), the weight of the complex operation is too big, the big-machine is strong but too simple (more than the half of its area is used for arithmetic and logic functions and less than half for control and communication), or the area used for arithmetic and logic functions in the small-machines is too big (the design of the small-machines is not efficient).
$\diamond$

When the supra-linear condition does apply Let be another case:
$R = 2;\ r = 3;\ q = 16;\ p = 5.$
In this condition a super-linear performance is possible.
$\diamond$

Therefore, in some conditions, multiplying the area N times improves the performance of the parallel machine more than N times, for a given application and/or with an appropriate design characterized by r (minimal circuits) and q (efficient algorithms for complex functions).

2.2 Applying the strategy

The strategy is applied for two cases. The first case uses as small-machines *byte-machines* (the arithmetic and logic unit performs simple operations on bytes). The second case uses *bit-machines* (the logic unit performs *any* operation on one-bit words).

2.2.1 Byte-machines

In all the following examples the *supra-linear condition* is verified.

How works R Let us emphasize first the effect of R (the weight of arithmetic and logic resources in the big-machines). The first case is of a machine strongly oriented on arithmetic applications, with simple control and communication.

1. $R = 2;\ r = 3;\ q = 16;\ p = 5.$
In Figure 4 line 1 represents $OPC(area)$ for a network of n small-machines. The super-linear performances occur for $N > 1$. Using the previous formulas results:

$n(2) = 9$
$n(3) = 15$
$OPC_{small\_machine} = 0.25$

The second case is of a more general purpose machine.

2. $R = 4;\ r = 3;\ q = 16;\ p = 5.$
In the same figure line 2 represents the performance of expanding the system in a network of n small-machines

$n(2) = 15$

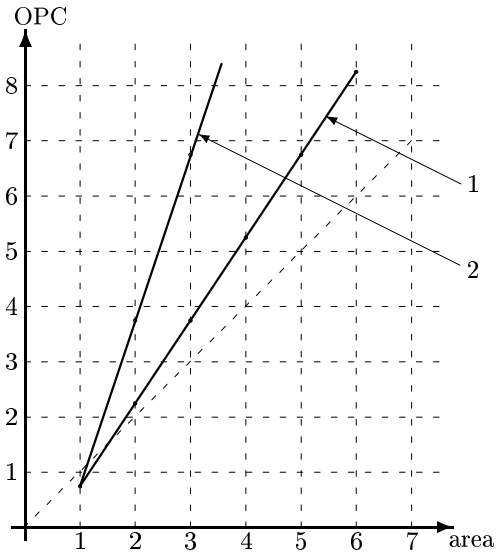

Figure 4: How is affected the performance by the weight, R, of arithmetic & logic in the big & complex machines. (1): big weight, (2): small weight.

$n(3) = 27$

We conclude that it is easier to improve the performance of a general purpose machine.

◇

How works q The effect of improving the algorithm will be exemplified (see Figure 5). In the first case the complex function is performed in 15 steps and in the second the algorithm is improved such that the complex function is performed in 11 clock cycles.

1. $R = 2; r = 3; q = 15; p = 3.$

$n(10) = 57$

$OPC_{small\_machine} = 0.17$

2. $R = 2; r = 3; q = 11; p = 3.$

$n(3) = 15$

$OPC_{small\_machine} = 0.28$

For a "good" algorithm an efficient machine starts from $n = 16$. For the bad case the same effect is obtained only starting from a network of $n = 64$ small-machines.

◇

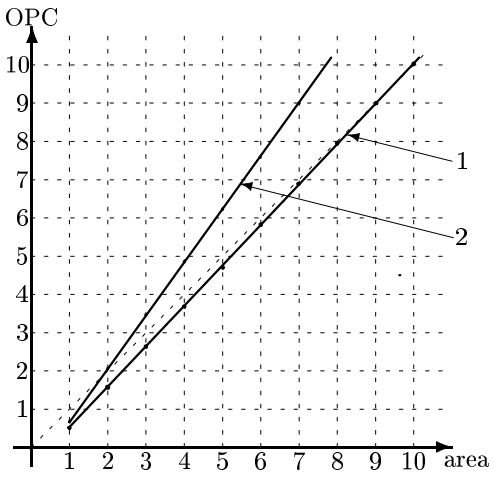

Figure 5: How is affected the performance by the algorithm which implements the complex operations (q). (1) for the low performant algorithm, (2) for the better algorithm.

How works r The effect of improving the design of small-machines will be exemplified. The second case is a "less inspired" design. The result is a smaller slope (see Figure 6) for the line describing the $OPC(area)$.

1. $R = 2; r = 3; q = 10; p = 3.$

$n(2) = 9$

$OPC_{small\_machine} = 0.25$

2. $R = 2; r = 2.2; q = 10; p = 3.$

$n(6) = 25$

◇

How works p The effect of changing the application domain will be exemplified.

1. $R = 2; r = 3; q = 10; p = 3.$

$n(2) = 9$

$OPC_{simple\_machine} = 0.25$

2. $R = 2; r = 3; q = 10; p = 2.$

$n(6) = 33$

$OPC_{simple\_machine} = 0.18$

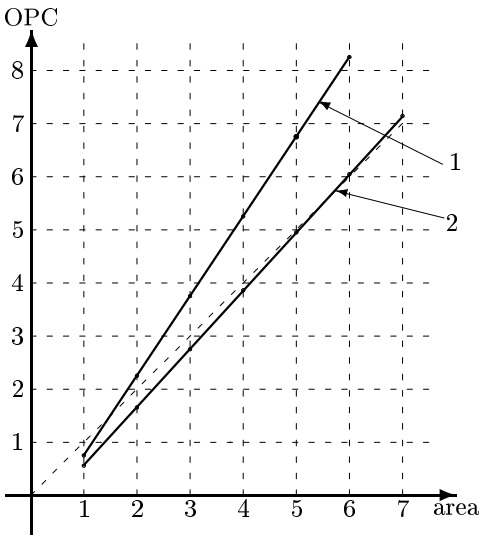

Figure 6: How is affected the performance by the quality of design (r) or by the weight of complex operations (p). (1) good design / low weight, (2) uninspired design / high weight.

The second case has less advantage. The result is a lower slope (see the same Figure 6) for the line describing the $OPC(area)$.

◇

2.2.2 Bit-machines

We expect for the smallest granularity to obtain efficiency for a bigger n.

How big is n for one-bit machines? Let us consider a network of very small small-machines defined by:
$R = 3;\ r = 100;\ q = 300;\ p = 1.2$
Results the dependency from Figure 7.

$n(5) = 1300$

$n(6) = 1600$

$OPC_{small\_machine} = 0.004$

◇

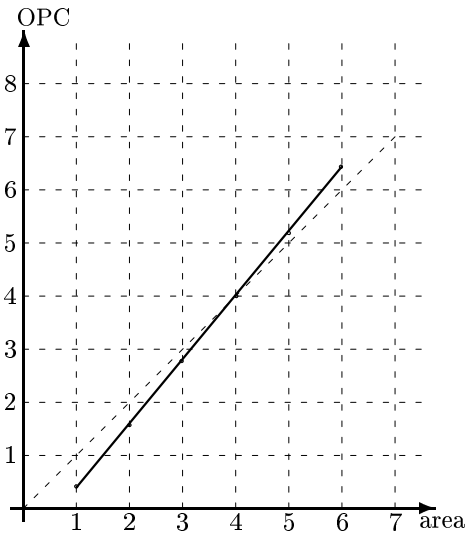

Figure 7: Performance for a network of one-bit machines.

2.3 Conclusion

For *byte-machines* enough computational power is obtained with an efficient structure starting from around $n = 64$.

For *bit-machines* enough computational power is obtained with an efficient structure starting from around $n = 2048$.

A demanding application domain (small p) can be satisfied only with a careful design (big r) and smart algorithms (small q).

3 Size vs. Complexity

The supra-linear performance hunted in the previous section pays mainly for:

- the increased area, whose price increases more than linearly
- the increased power needed for connecting subsystems on a larger area
- the difficulties to maintain a high speed clock on a big chip or in a multi-chip/multi-package solution.

The proposed strategy can be developed taking into account how the price of area evolves, and the type of interconnections used by the system.

The main consequence of "simplicity" in the highly granular systems consists in the intense use of the structure in each clock cycle. Indeed, composing complex functions starting from simpler ones means to use continuously the entire hardware. In a complex big-machine, for example, a multiply unit is used only for executing multiplication. In a simple small-machine, its simple arithmetic and logic unit is continuously involved in processing.

> **The basic rule of big (parallel) systems**: *the complexity of a large system must grow slower than its size.*

The question arises: *how is able a simple system to perform complex computations with high and very high performances?*

The answer: *fortunately*, computation is simple because:

- a program loop can be converted into a repetitive physical structure
- *almost all* compositions are used to define primitive recursive functions
- any primitive recursive definition (which is inherent sequential) can be transformed into an iteration (involving pipeline parallelism)

The main limit: the restrictions imposed by the partial recursive functions. Which means pure parallel computation can not be performed.

The *good news* is: the huge majority of the computation in real signal processing are primitive recursive.

We are helped by the strange fact that even if only a small minority of computable functions are primitive recursive, the human being is obstinated to use them almost all the time.

References

[1] Gregory Chaitin: "Algorithmic Information Theory", in *IBM J. Res. Develop.*, Iulie, 1977.

[2] Daniel Hillis: *The Connection Machine*, The MIT Press, Cambridge, Mass., 1985.