Boundaries of Supercomputing

NP Revisited

Lutz Schubert and Stefan Wesner
HLRS
University of Stuttgart
Stuttgart, Germany
{schubert; wesner}@hlrs.de

***Abstract*—Supercomputers can reach an unprecedented degree of scale and miniaturization reaches the quantum level of manufacturing. Non-regarding this progress, however, computing capabilities are and will remain insufficient to meet the demands of many compute intensive scenarios. The major obstacle thereby consists in the non-deterministic polynomial (NP) nature of these problems. Recent research and development seems to have almost forgotten about this intrinsic problem. With this paper we want to remind of the relevance of NP for supercomputing by exploring its impact on future computing development and discuss potential approaches to relieving (not solving) this issue.**

Keywords- high performance computing, NP, scalability, non-determinism, parallel computing

I. INTRODUCTION

The number of computational units that are available to a user increased constantly over the recent years: super-computing clusters integrate thousands of processors with high speed interconnects that allow the user to execute large scaling applications. Multicore processors bring parallelism to the common desktop PC and even though scale still ranges in the area of 4 to 8 cores, manufacturers already plan on processors integrating 100s of compute units, so that today's cluster scale will be available for desktop machines in 10 to 15 years' time. In addition, the resources available over the internet and thus principally available for grid, cloud and P2P computing have reached several millions by now [1].

Even though the effective global computational power is high, there still remains a large set of problems that cannot be solved – this includes accurate (long-term) weather forecasting, simulation of the human being, astrophysics etc. All approaches so far provide approximations rather than accurate results – mostly this is due to the size of the respective system, which in the case of weather forecasting and astrophysics is "open", meaning that a potentially infinite number of parameters impact on the computation - classically, this is referred to as dealing with "LaPlace's demon" [2]. It is obvious that "open world" problems are unsolvable due to the physical limitations of the resources – however, most of these problems can be reduced to a subspace in which parameters have only minimal impact (e.g. the gravitational forces across large distances) and thus can be subsumed to a simpler factor or even neglected.

As we will show, the actual main computational problem however consists in the non-determinism of the respective systems, i.e. their "chaotic" nature. Non-regarding the wording, this does not imply that the computation is not causal, but that there is no functional representation of the results for any time t - instead g(t+n) can only be generated by (n) stepwise iterations from g(t). Computation of g typically involves multiple iterations for approximation, which means that the complexity for calculation quickly increases beyond the capacities of existing infrastructures and, in fact, will always exceed these restrictions (chapter II).

All current development concentrates on increasing the number of computational resources by exploiting parallelism – be that on the level of the instructions or on the level of the full processing unit, or on increasing the execution speed by employing specialized accelerators. In all cases, the capabilities effectively increase polynomial whilst the requirement growth remains exponential. In chapter III we will elaborate why such development is insufficient.

This restriction is due to the fact that modern computing still builds on Turing's model, which is strictly sequential in nature. In order to cope with the NP class of problems, a new computing model is required which can deal with the non-deterministic nature of these tasks. In chapter IV, we will discuss what such a model could look like.

We conclude the paper with a discussion on the obstacles towards realizing such a computing model.

II. NP APPLICATIONS / NON-DETERMINISM

Simulating real world behavior is essential for both academia and industry: not only to understand the mechanics of the system examined, but in particular to be able to modify or replicate it. Thus enabling for example the evaluation of a design prior to its production and to estimate (and contain) impact on the environment, such as oil leakage, air poisoning etc. This equally affects all disciplines, ranging from engineering over natural sciences to social studies.

These disciplines typically investigate different levels of the system, from subatomic (quantum physics) over living creatures (biology, sociology etc.) to galaxies and beyond (astrophysics). And even though the range seems well defined for most disciplines, such as medicine, ranging from individual cells to living beings, there is nonetheless an important reciprocation between most of these levels. This interdependency sometimes leads to the emergence of new, merged disciplines, such as biochemistry, and in other cases one of the major concerns consists in eliminating all influence from other systems, such as in quantum physics. It becomes more and more apparent that effectively all levels have to be considered for accurate predictions and simulation. For example the virtual physiological human (VPH) community aims at simulating the whole human body on all levels (i.e. from cell systems down to molecules) in order to predict e.g. the spread of medicine in the body.

The interest in such research is due to the cross-impact of other domains unto calculations of the respective system. It is a specific aspect of natural systems that they are essentially chaotic, meaning that miniscule changes in parameters lead to completely different results. In other words, minor errors in the data can lead to completely wrong results. Since natural systems are also mostly "open", there is an infinite number of impact factors. For example, in order to measure an exact kilogram, already the gravity shift due the planetary constellation plays a crucial role. The impact of the combined two factors – openness and chaotic – is also well known as the so-called "butterfly effect" [3].

There is no direct determinism in the underlying functionality, that allows calculation of f(t) for any t directly. This is simply due to this large degree of interactions between all parameters, leaving a potentially infinite number of equations to be solved in each iteration step. Whilst the scope can be reduced according to the number of particles considered in the subspace and according to the strength of coupling, it still leaves the system essentially unsolvable without approximation and optimization.

A. *The Impact Of NP: An Example Application*

Let us examine this in a simple example of particle collision in an isolated subspace, where each particle can be simulated as (ideal) snooker balls: even though each trajectory can be represented as a vector, collisions need to be checked with all other particles (leaving mechanisms for reducing the search space aside). In the most straight forward approach, we would therefore advance the position of each particle per time step by a safe distance thereby checking with all other particles whether a collision would occur in the respective time step, and reflect it accordingly.

Since at any time a collision may occur (or even several at once), the outcome at any time t depends on the constellation at t-1. In other words $f_n(t+1) = g(f_n(t))$, whereas $f_n(0)$ is the initial constellation and g(x) is the combined collision test and movement over n particles. This means that there are n! equations to be solved in g(x) for one time step.

If each test could be performed by an individual computer in order to achieve maximum performance, adding only one particle more would require n*n! more resources. This can be simply shown: if the complexity for calculating $f_n(t)$ is n!, then the complexity of $f_n+1(t)$ is (n+1)!. Therefore:

$$(n+1)! = n!*(n+1). \tag{1}$$

To be more concrete: for 1.000.000 (10^6) particles, $8,26*10^{5.565.708}$ resources would be required. Just adding one particle to this would lead to additional(!) $8,26*10^{5.565.708+6}$ resources. In one cubic meter of air alone there are more than 10^{25} molecules and hence particles to be calculated. There are multiple methods to reduce the number of calculations, such as neighborhood restrictions etc. The main point of this example is not so much to show the complexity of the calculation itself, but the enormous growth of requirements with only slight increments in the problem or data space.

The usual approach to dealing with this amount of equations consists in approximation and result estimation. Chaotic non-deterministic systems can however lead to enormous result deviation if just a single value is changed minimally. In the example of the particle system, we can see how errors can sum up over time, assuming that just a single particle shows a vector deviation (speed or angle) of ε between simulation and real world. We can calculate the average time t_{col} for a collision between two particles to be

$$t_{col} = V / (N\pi D^2 v) \tag{2}$$

with V being the volume of the subspace, N the amount of particles within V, D the diameter of the particles and v the velocity. For the sake of simplicity, we assume that all particles have a diameter of 1 angstrom (nitrogen has 1.5 angstrom, oxygen 3.6). With a density of 10^{25} molecules/m^3 and the average velocity per particle of 500 m/s (air has roughly 463 m/s at 20° C), this leads to roughly $157*10^6$ collisions per particle and second (for air, this is roughly $5*10^9$).

Ideal elastic collisions preserve the energy of both particles – i.e. given initial vectors $\overline{v}_{1o}$ and $\overline{v}_{2o}$ of two particles, the new vectors $\overline{v}_{1n}$ and $\overline{v}_{2n}$ sum up to the same combined vector. Without going into full detail, one can show that an initial error ε of just one particle, i.e. $\overline{v}_{1real} = \varepsilon * \overline{v}_{1simulated}$ is maintained across all collisions and even transplanted onto all colliding particles [4]. Due to the exponential nature of the collisions, the total (maximum) error after one second is $\varepsilon*2^{157*10*6} \approx \varepsilon*5*10^{2835}$ - for reasons of simplification, we ignore the cancellation of two errors, so that the total error will be slightly lower.

Notably, this does not hold equally true for all problem fields (see section V). Since the calculations themselves are just approximations the effective error is essentially higher. Accordingly, one of the major efforts consists in keeping ε as small as possible. However, limitation of resources and exponential growth of complexity is compensated on cost of precision, thus leading to higher, rather than lower ε and thus to less precise results.

B. *Dealing With NP Applications*

There are multiple ways to address problems with exponential complexity – typically these consist in restricting the problem space, subsuming multiple equations under one, reducing the precision, using approximation calculations etc. Given the potential error introduced through these methods, the major interest is obviously to employ means that are more precise, thus reducing the risk of an increasing error.

Much effort is vested into finding a representation of the according task in the polynomial problem space, in other words to reduce the specific element of NP to a respective representation in P which would reduce the resource need and complexity by an exponential factor. HPC programmers spend much effort into finding such a reduction, yet that effort is exponential in itself. Even though there is a set of problems which can clearly not be reduced to P, how much of the NP space is identical to P is unsolved as yet [5]. Ideally, the set of NP problems equals the set of P, in which case all major mathematical problems could be calculated in polynomial time.

In addition to "ordinary" NP problems, there is a set of problems which cannot be reduced to P, generally referred to as "NP-hard". Any problem belonging to this space will exponentially grow in complexity with the degree of desired granularity, respectively data size. In these cases, desired accuracy must be weighed against computational effort. Higher accuracy and larger data set are nonetheless urgent demands from industrial and academic research.

III. THE DEFICIENCIES OF CURRENT DEVELOPMENT

The classical approach to increasing the performance of a processing unit consists in increasing its execution clock rate, e.g. by higher settling rates or extended instruction level parallelism allowing for execution of more operations at once [6]. However, these approaches face multiple problems and effectively do not really improve performance any more [7]. Much optimization is nowadays handled by the compiler, rather than the hardware, even though specialized processing units that offer application specific optimization capabilities are growing in interest (see below).

Current manufacturers extend parallelism on a higher level by exploiting principles that have long been employed in high performance computing: parallel processing units. As opposed to ILP, parallelization on this level implies effectively complete replication of the whole processor, including the logical unit, cache and I/O. Modern multicore processors are thus exactly that: multiple full units within a confined space, though the architecture of interconnects etc. has slightly changed in order to maximize performance.

A. The Limitations of Scale (or Why Multicores Are Not The New Messiah)

Multicores thereby face the same issues as cluster computing: applications simply do not scale to the amount of available resources. In other words, we already have access to more compute units than most programs can effectively use. This is due to two major constraining factors: Amdahl's law and messaging overhead.

Amdahl's law generally states that the speedup of parallel program execution is limited by its sequential aspects, or in other words: there are functions and code segments in any program that simply cannot be parallelized. This statement was later turned into a general formula for speedup:

$$\text{speedup} = 1 / (r_s + r_p/n) \qquad (3)$$

whereas r_s denotes the sequential and r_p the parallel portion of a program and n the number of parallel processes, i.e. compute units. Plotting this function clearly reveals how the speedup gained by parallelization saturates with a specific number of compute units (cf. Figure 1.)

It is notable that only applications with a very high degree of scalability (> 90%) can actually make use of the number of processing units offered in large scale clusters and even there, the actual gain is comparatively low. This calculation however does not even consider the impact of messaging overhead or the sequential properties of the individual processes themselves, let alone the ratio between messaging and workload of the processes.

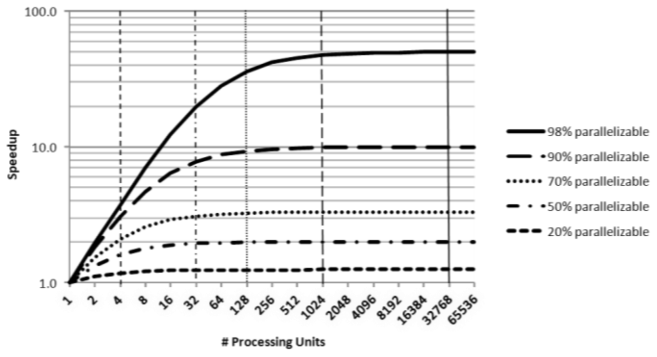

Figure 1. Speedup of an application according to Amdahl's law

Messaging Overhead thereby is the major problem for all parallel programs – this includes data exchange between processes as much as access to (remote) resources, such as memory or hard drive. Whilst access to memory is specifically defined by the limitations of cache per processing unit, the impact of data exchange between processes is particularly dependent on the distribution of tasks and / or data across processing units:

Let us assume that a given task consists of n iterations (per value) on a dataset with m values. In a straightforward algorithm this means that n*m iterations have to be processed. Assuming that a single processor would take t_{exec} seconds to execute this task, p_{total} processors would take t_{exec}/p_{total} seconds without overhead for distribution and synchronization. In the ideal case, we have n*m resources available, each thus only processing one iteration for one datum. Leaving aside the fact that data needs to be passed between iterations, this distribution is only sensible if the time for gathering the results t_{msg} is higher than the time for execution of one iteration $t_{exec}/(n*m)$. Otherwise, $p_{ideal}=[(n*m)*t_{msg}]/t_{exec}$ defines the number of iterations per processing unit that should at least be executed in order to not reduce performance through messaging – just for result gathering, i.e. for embarrassingly parallel tasks.

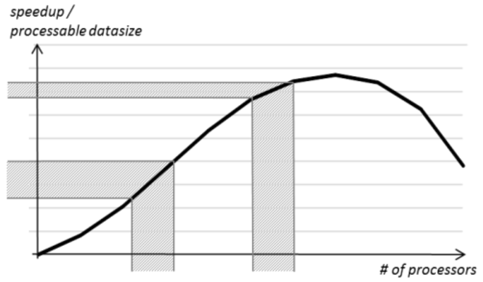

Figure 2. Speedup of program parallelization in relationship to the overhead produced by messaging for synchronization purposes

In real cases, the degree of messaging is obviously defined by the dependency between and across iterations, i.e. which data is required for a single iteration and which is carried across. Typically, a kind of synchronization step is required at least once per iteration. Therefore if the execution of a single iteration becomes too small, messaging produces more load than process execution itself.

Without specifying a concrete unit, as this depends on the respective use case, we can therefore say that performance increases with parallelization up to the point where the messaging overhead exceeds the optimal ratio of workload to messaging of an individual processor. This leads to a effective speedup figure as depicted in Figure 2.

In fact few applications scale well over a few dozen processing units and the best scaling applications are denoted by very little communication. This is particularly true for embarrassingly parallel jobs, such as rendering.

B. *Not Enough Resources?*

Performance of parallel applications and hence usefulness of large scale supercomputer is naturally limited, basing on the type of calculation to be performed. But it is exactly this type of highly dependent calculations that need to be executed with higher accuracy and over larger datasets in order to satisfy industrial and academic needs. By increasing the workload per processor, by improving the interconnect and by reducing the sequential portion of the program, this saturation point can be pushed to higher scalability numbers - however, an even more common approach consists in combining different levels of process interdependency by linking simulations of different scale [8]. This is in principle identical to segmenting the problem space into sub spaces, thus improving data correctness on a large scale, i.e. across the individual simulations' boundaries, but not within the given segment.

Even if manufacturers could reduce the interconnect problem and the sequential workload, problems with exponential complexity growth would still exceed the number of available resources. As noted, current manufacturers all aim at increasing the number of resources rather than increasing the performance of the individual processing unit – however, the effective gain of this approach decreases with the number of resources, as the amount of data that can be processed in a given timeframe is in direct relationship to the performance of the system. One can interpret Figure 2. also as the process-able size of the dataset over the amount of cores: it can be clearly seen that an increase in the amount of cores in small scale processing systems leads to a stronger increase than in larger scaled ones (gray boxes in the figure).

As opposed to this, increase in clock rate leads to a uniform increment in data-size that can be processed by factor n. More concretely, an increment of the clock rate by factor c also increases the amount of process-able data by c. However, due to the power wall issue, manufacturers must decrease the clock rate when integrating more compute units into a single processor [9] – the main problem for manufacturers is therefore to find the best relationship between amount of cores and clock rate of the individual units. As this relationship is strongly application dependent, there is no clear solution as yet.

Leaving aside the effect of messaging overhead, jitter, limitations of scale etc. and assuming an ideal scalability performance, i.e. where the combined performance is defined over the sum of all processing units' clock-rates:

$$p_{comb} = \Sigma_n p_n \tag{4}$$

(with n being the number of processing units and p_n the respective performance / clockrate), we can easily show that the effective (combined) performance in current systems does not grow according to Moore's law anymore. Instead the growth has effectively decreased from exponential to linear. Mapping this to the complexity to size ratio (Figure 3.), it is obvious that as we advance the performance of computing systems basically linear (following classical mechanisms), the complexity these systems can handle grows linear too. Implicitly, the size of the according problem space grows only logarithmically.

To summarize:

- the number of necessary resources grows exponentially to the complexity in NP problems
- the performance increase through current large scale systems is naturally limited
- most applications do not meet the scaling capabilities of the underlying hardware

IV. NP PROCESSING

As the interest in more accurate processing of larger data sets increases, so does the pressure on computer manufacturers and application developers to deal with large scale. However, as can be clearly seen from the discussion above, the growth of resources needed exceeds the capabilities exponentially over time – in other words, whilst manufacturers and developers try to push the degree of scalability further up the scale, the need for scale and efficient usage thereof grows faster than manufacturers and developers can achieve. As all the major improvement steps have been taken, the impact of all the minor adjustments that can still be taken in order to increase scalability constantly decreases and becomes more and more use case dependent.

One reason for this deficiency is caused by the limitations of current processors in terms of dealing with non-deterministic problems. More specifically, the strict "sequentiality" and determinism of the Turing machine prevents current computer models to deal with NP problems and the implicit communication. Given the effort to replace a current, well-established computing model with a potentially non-interoperable alternative model, the major question to pose is: what would be the benefit of having machine that can deal (better) with NP?

A. *The Impact of Reducing NP to P*

As noted above, the hard task for HPC developers consists in finding a representation of the NP problem in the polynomial time space, or at least an approximation. One principle thereby consists in subsuming multiple equations into more global, general equations that, even though they disrespect e.g. interactions between particles in the subspace, still deliver results accurate enough for the purposes of the task. The actual error may be quite substantial in such an environment, where not only the individual deviations sum up, but also the additional error for aggregation and approximation of multiple equations contribute to the overall deviation. In other words the result becomes inaccurate by the factor of potential deviation as calculated above.

If, however, an accurate representation of the respective problem in P space can be found, the error (and hence accuracy) of the result is maintained, i.e. not increased further by the according subsumation and approximation. Since many problems in NP actually belong to the P space, this is frequently possible, though very difficult to achieve. There is furthermore no proof whether all NP problems can thus be represented as P tasks.

Nonetheless, the gain achieved by this complexity reduction is obvious. Figure 3. depicts the complexity reduction and hence the decrease in time to complete the task. It can be noted that for small n (and small c), the complexity of NP problems is actually lower than that of polynomial tasks – this however is quickly surpassed (note the logarithmic scale) with growing n.

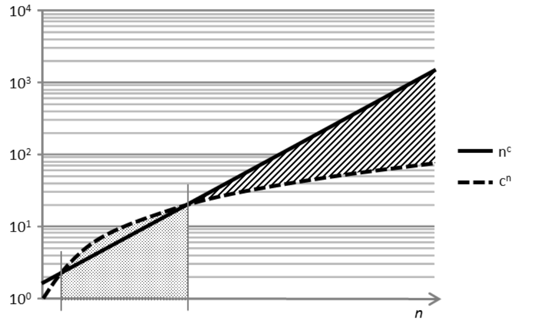

Figure 3. The complexity of n^c versus c^n. The striped area denotes the overhead of NP over P. In the left area, P exceeds NP in complexity

In our particle collision example, the complexity of the original function is n!. Reducing this to a set of equations of complexity n^c would reduce the complexity by factor

$$n! / n^c = (n-1)! / n^{c-1} \tag{5}$$

Whereby we must assume that c is comparatively high, so that an improvement is notable only for large datasets (cf. Figure 3.). However, as noted, it is not always possible, let alone easy to find a P representation for an NP problem, so that ideally, the task is approached the other way round:

B. *Classical Approaches to NP Computing*

Essentially, if the processing unit itself can process non-deterministic tasks, an NP task on top of it will essentially be executed in order of complexity P. Obviously, this is easier said than done, as otherwise NP processors would have long since emerged on the market. Nonetheless it needs to be stressed again, that the according industrial interest has simply not arisen so far – instead primary focus rested on advancing existing computing types, i.e. Turing machines.

Back in the seventies and eighties, some attempts have been made to instigate "unconventional" processors and in particular to examine the capabilities for NP processing via other means, such as biochemical computing etc. The lacking success is thereby less due to the lack of quality of the results, but rather again for the lacking interest from industrial side: switching from the successful Turing model to an architecture that has neither proven successful nor can be easily applied to common problems would imply too many manufacturing costs and risks.

Many of the according approaches based on the increased scalability of the system rather than the respective capabilities to deal with non-determinism. Reduction in size in comparison to electronic PCs often provokes the mis-conception that the according capability to deal with NP problems is higher. This builds on the same mis-understanding as the assumption that multicore processors and large scale systems can solve the resource need of the NP space – as has been shown, this is not the case though.

On a similar basis, it is often assumed that quantum computing (QC) would essentially enable non-deterministic computing, and thus solve the NP issue. However, QCs like normal desktop PCs are purely deterministic and sequential in their processing. Even though quantum processes are non-deterministic in nature, the according effects are not exploited in QCs. Miniaturisation and interconnectivity reaches a peak in QC, thus allowing for enormous scale and in theory, quantum processers could perform calculations over full real time numbers as opposed to pure bitwise operations on PCs – however, this faces the same obstacles as analog computing did during the 60ies and 70ies, and is unlikely to be successful due to the same issues [10]. It must be expected though that quantum computers (if ever realized) represents one of the upmost boundaries of scalability or rather of miniaturization.

C. *Alternative Paths to Computing*

The main reason for this failure to cope with NP consists in our restricted way of thinking in terms of computation, which is still essentially Turing in nature. Processes in nature are therefore examined for how they can be converted into Turing machines, not how they functionally behave. In other words, the processes are interpreted deterministically, without actually exploiting their non-deterministic nature.

Alternative paths to computing must therefore focus on exactly this rather than forcing determinism onto these processes. Instead of exploiting particle collision for message transaction, it can instead be considered as a segment in a chain of non-deterministic events that can be expressed as particle collisions. In the simplest case, a well-defined sub space of particles can simulate the overall behavior of aerodynamics in a larger space etc. Only few approaches try to address the computational nature, in the sense of the underlying processing logic:

A spin-off of MIT for example investigates into pro-cessors that replace the underlying binary logic with a probabilistic logic [11]. This does not address non-determinism in the actual processing, yet it allows for more efficient computation of all probabilistic problems, as involved in most NP tasks.

Some natural systems are not only non-deterministic, but are capable of dealing with it. These involve swarms, neural networks and other self-organising systems which are capable of solving problems, such as finding shortest routes towards a food source etc., which belong to NP and lead to high complexity when simulated. Such systems effectively employ mechanisms of dealing with imprecision and fast adaptation in order to approximate a locally optimal solution. A certain degree of scale allows improving optimality further through redundancy, where by it must be noted that the scale is essentially polynomial to the complexity.

Also, the principles of molecular computing are capable of dealing with bounded NP problems in a polynomial number of steps and a limited size of scale [12].

In such systems the problem (and hence the goal) is either implicitly encoded, such as finding the food source for a swarm system, or needs to be painstakingly trained, such as for neural networks. In both cases it generally limits the system to the specific problem domain. The effort of "coding" the problem is hence in most cases NP itself which considerably restricts its applicability.

V. Conclusions: Next Computing Models?

Complexity is a growing problem for computing, that is often ignored or considered to be a pure problem of scale, i.e. that can be overcome by future large scale computing systems. However, this is building on the – generally wrong – assumption that problem complexity scales linear (or polynomial) with the problem space and that performance grows linear and unbounded with the amount of computational resources. Whilst the NP problem space and its implications are actually well known, the actual consequences of it are often ignored or simply forgotten.

The seeming unboundedness of performance through scale however only arises from the fact that many supercomputing problems still move within the area of a positive scale to performance ratio (left gray area in Figure 2.) in the last years, but reaching its peak (right gray area). Personal computers on the other hand just only have reached the beginning of the scale to performance curve, so that still considerable improvements can be achieved through increasing the amount of computational resources [13].

Investing in scale rather than in complete new computing systems is also economically more viable and less disruptive in the short run. As we reach the peak of performance, such a disruptive paradigm switch will become imminent. Current approaches to improving the performance over scale, in particular by reducing the impact of messaging or exploiting more concurrency / asynchronicity, and even quantum computing will only help delaying this problem, i.e. stretching the scale to performance ratio. The main problem can however not be overcome this way, as messaging will always create delay in execution with a certain point of scale ("speed of light is not fast enough" [14]) and the resource need of NP problems grows exponentially and unbounded. Nonetheless, first attempts in that direction need to be seriously undertaken within the near future in order to compensate for the delay in research and development, until more long-term results have been achieved.

That some systems principally can deal better with the NP problem space has already been shown through attempts already initiated back in the 1970ies and 1980ies which base in particular on self-adaptation under uncertain conditions. But also stochastic mechanisms, combinatorial optimization, elastic scale, bounded non-determinism, dynamic segmentation etc. all have introduced principles that significantly contribute to the capability of dealing better with the complexity of such problems and reducing the impact of NP. This however is far from maturity as yet.

Acknowledgment

This work was partially supported by the S(o)OS project under FP7-ICT-2009.8.1, grant agreement no. 248465. See www.soos-project.eu for more information

References

[1] Netcraft: May 2010 Web Server Survey. (2010). http://news.netcraft.com/archives/category/web-server-survey/ [accessed: 2012-06-06]

[2] Laplace, M.D.P.S.: A Philosophical Essay on Probabilities. Forgotten Books (2009)

[3] Lorenz, E.: Atmospheric predictability as revealed by naturally occurring analogues. In: Journal of the Atmospheric Sciences 26: 636–646 (1969).

[4] Leckie, W., Greenspan, M.A.: Pool Physics Simulation by Event Prediction 2: Collisions. In: ICGA Journal 29(1): 24-31 (2006)

[5] Richard, L.: On the Structure of Polynomial Time Reducibility. In: Journal of the ACM (JACM) 22 (1): 155–171. (1975)

[6] Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann (2006).

[7] Manferdelli, J.: The Many-Core Inflection Point for Mass Market Computer Systems. In: CTWatch Quarterly 3(1) (2007). http://www.ctwatch.org/quarterly/articles/2007/02/the-many-core-inflection-point-for-mass-market-computer-systems/ [accessed: 2012-06-28]

[8] Hox, J.J.: Multilevel analysis: techniques and applications. Routledge (2002)

[9] Jones, B.L.: Do Newer Processors Equate to Slower Applications? DevX online publication (2010). Available at: http://www.devx.com/enterprise/Article/34588/1954 [accessed: 2012-06-12]

[10] Aaronson, S.: The Limits of Quantum Computers. In: Scientific American 3 (2008)

[11] Feldman, M.: Startup Aims to Transform Computing with Probability Processing. HPC Wire (2010). Available at http://www.hpcwire.com/hpcwire/2010-08-16/startup_aims_to_transform_computing_with_probability_processing.html [accessed: 2012-04-18]

[12] Beigel, R., Fu, B.: Molecular Computing, Bounded Nondeterminism and Efficient Recursion

[13] Wesner, S., Schubert, L. Kuper, J., Baaij, C.: Convergence of HPC and PC: Really? In: Proceedings of the UK e-Science All Hands Meeting 2010 (2010).

[14] Alessandrini, V.: DEISA Perspectives - Towards cooperative extreme computing in Europe. Fourth EGEE Conference (2005).