Discussion on the Future Hardware

Alfio Lazzaro
CERN openlab

Workshop on Concurrency in the many-core Era
Fermilab
November 22nd, 2011

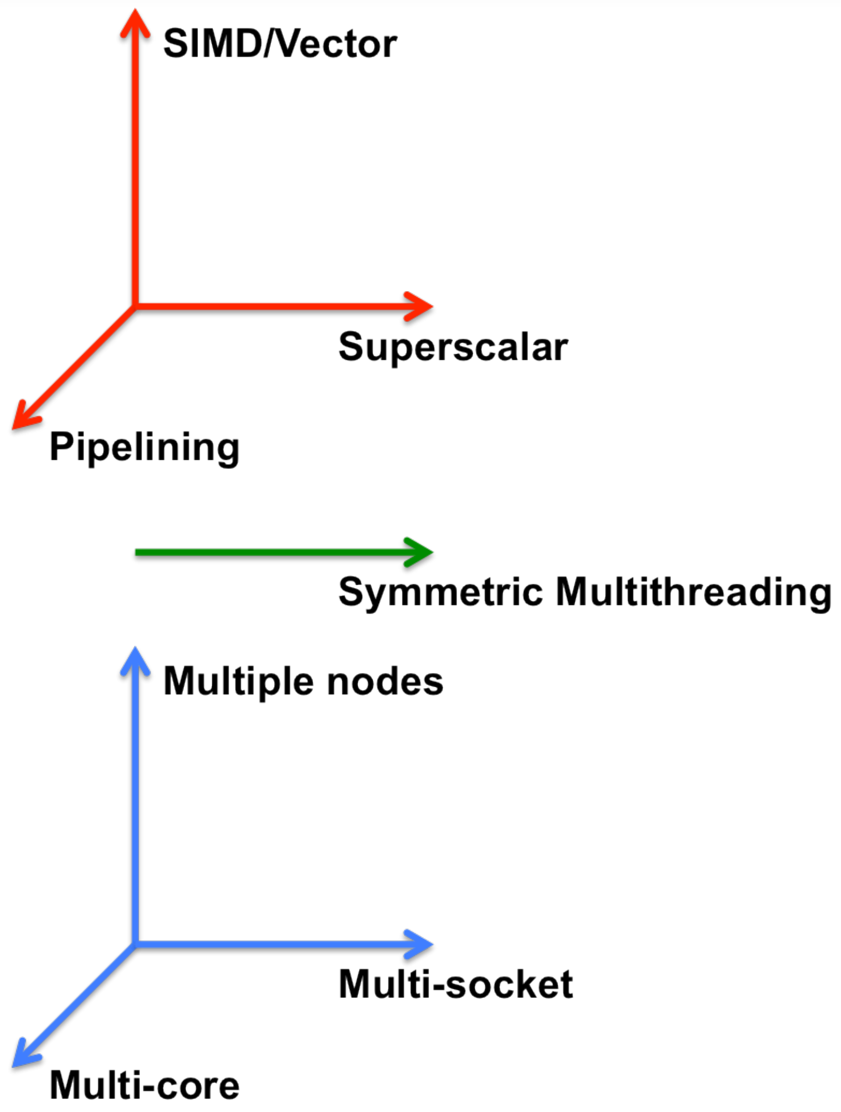
Why parallelize?

- Although parallelism is everywhere in the hardware, it is not clear (to me) why the current model of “embarrassing” parallelism would not scale in the future
  - Strong reason: reduce memory footprint?
  - Increase efficiency? (should we consider it a kind of optimization?)
- Can we gain in performance?
  - I don’t think we are interested in strong scaling (go faster)…
  - We are more throughput-oriented, i.e. weak scaling (definitely no HPC)
- Side effects? A lot…
  - Refactoring large portion of the code
  - Maintenance large parallel code (no trivial parallelism, like in HPC)
  - Several level of parallelism, require expertise in the community (hidden parallelism?)
  - Reproducibility can be an issue…
- The situation is evolving very rapidly. Should we wait?

Hardware direction

- Assuming that we are interested in x86-like CPUs, we can consider three big families:
  - “Fat” cores
    - Increase performance on a single core (Sandy Bridge gives ~10% more performance for the same frequency)
    - More and more cores will be available (8 cores in the Sandy Bridge @ 32nm, 10 in the next generation (2014?),…)
    - Keep cache coherency
  - “Light” cores, e.g. mobile devices (Atom CPUs)
    - Here ARM plays a major role though…
    - Not clear roadmap from Intel
  - Specialized cores (accelerators), i.e. Intel MIC and GPUs
    - Getting FLOPS more efficiently (at least 3x better)
    - Target specific cases, but several limitations on the software and hardware
    - Vendors are putting a lot of efforts on this sector. We can expect a lot of developments in the short

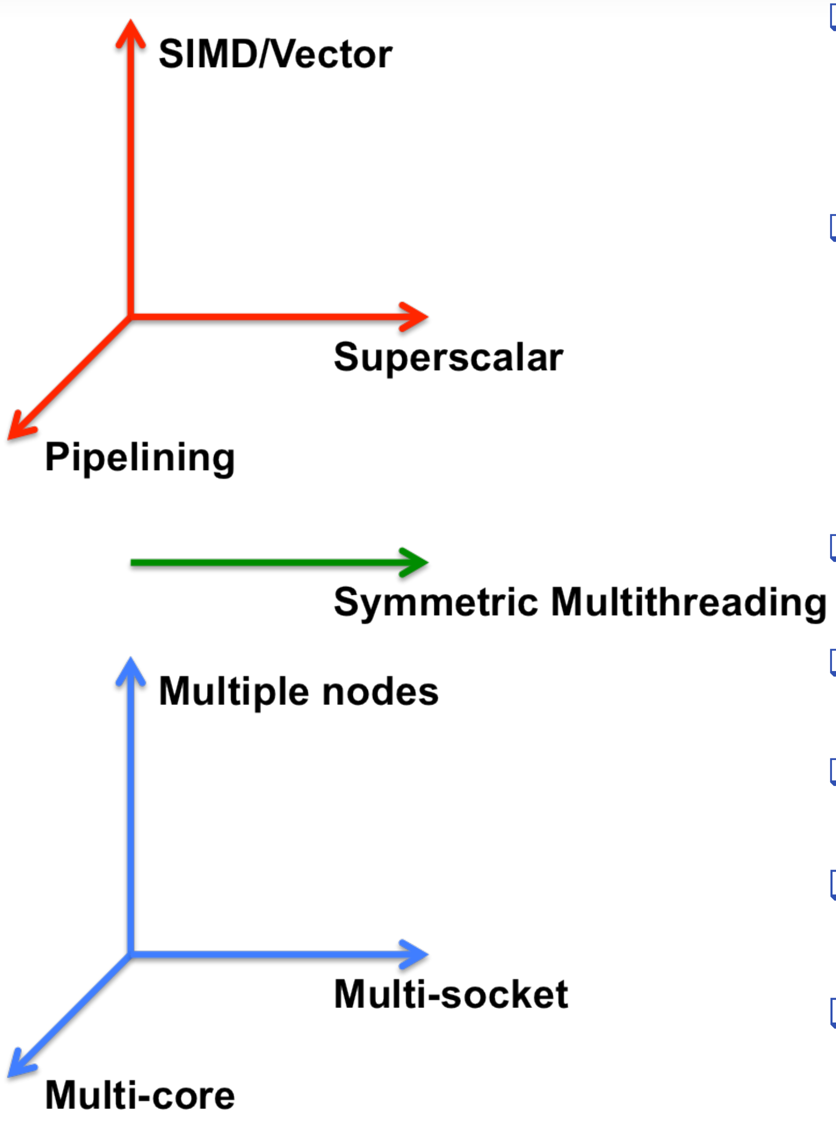
7 dimensions: Software



From Sverre Jarp presentation

- Look for improving everything
- Situation
  - We don't use Vectors
    - Not easy to use in our code
  - We don't use Superscalar
    - ~0.5 instruction per cycle (4 is the maximum)
  - Pipelining easy to break in complex C++ applications
  - We don't use SMT (2 threads per core, test shows +20% performance)
  - We would like to use Multi-core and Multi-socket
    - Cache and memory access (NUMA) problems
    - Thread safety
    - …
  - We are not interested in multiple nodes parallelism (a la MPI)

7 dimensions: Hardware

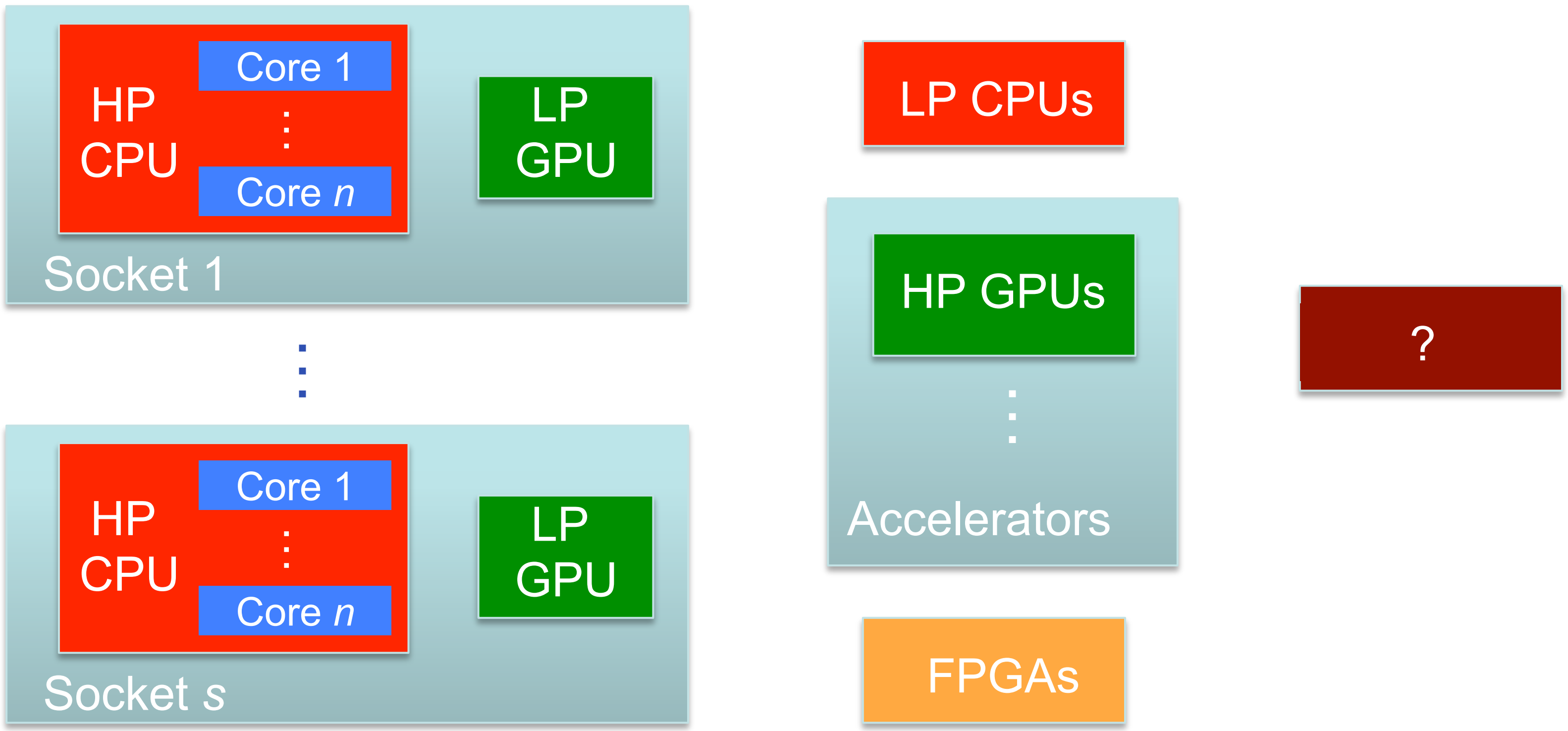


From Sverre Jarp presentation

- Increase in dimension of vectors:
  - AVX, 256 bits (4 doubles SIMD)
  - MIC, 512 bits
  - Expected to increase in the future
- SMT can be an efficient solution to hidden latency to memory
  - 2 threads on XEON
  - 4 threads on MIC
  - Several (>10) on GPUs
  - Can increase (?)
- More and more cores, sharing part of the resources on the chip
- Multi-socket
  - 2 or 4, can increase
- Multiple nodes on the same chassis
  - Micro-server: several independent nodes
- Accelerators can be considered a new dimension
- An interesting feature is the Turbo mode (Intel and now AMD):
  - increase clock frequency when not full loaded
  - Reduce Amdhal's law effect!

Looking in the future: Heterogeneous systems

- All systems give the best performance for specific tasks
  - There is not a unique system which is suitable for everything!
- It is a common understanding that future systems for computation will be a "heterogeneous" systems, where each sub-system will properly perform its part of execution



HP = High Performance; LP = Low Performance

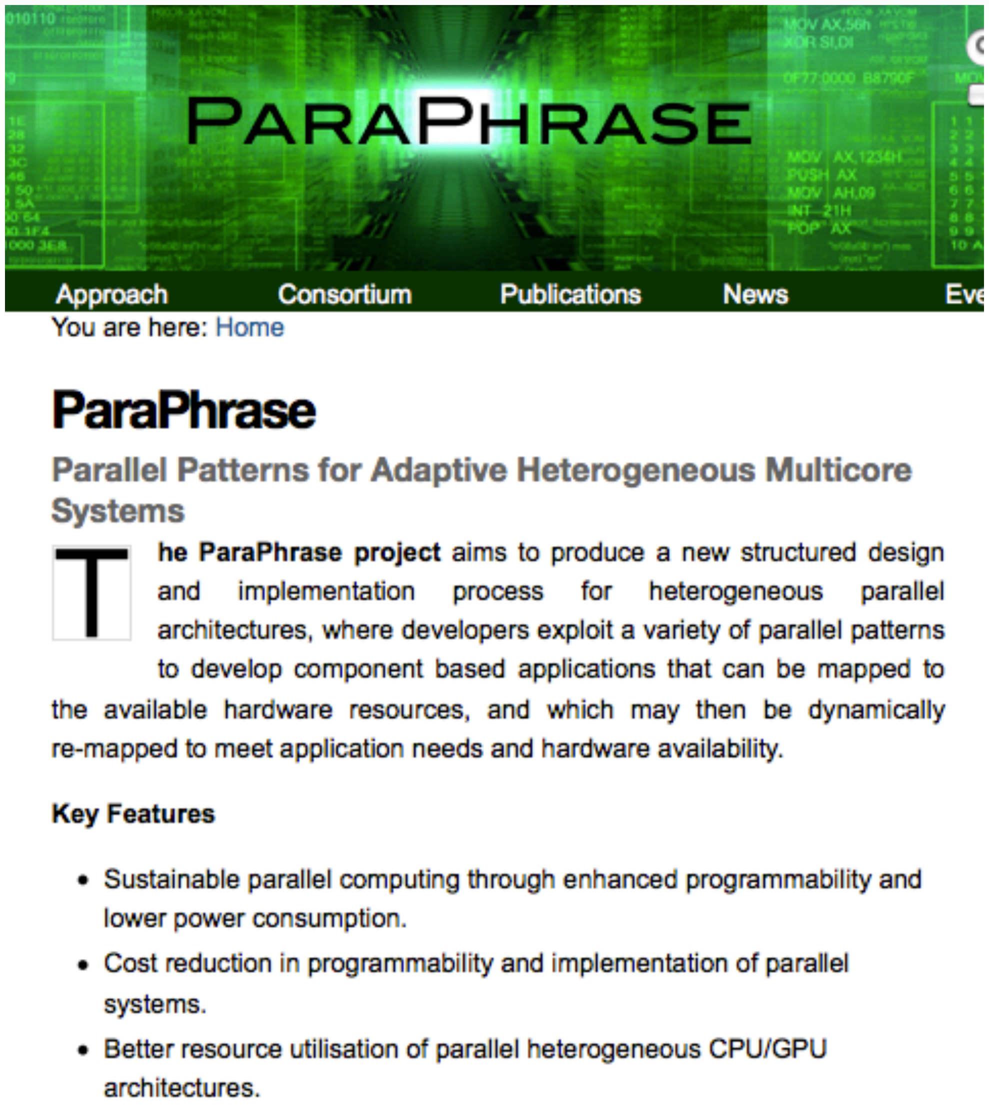
Looking in the future

- Depends how far we go…
- At least for other 5 (?) years the number of cores and the overall performance per chip will increase
  - However this model cannot give us exascale systems
- Accelerators are expected to play a major role
  - Scaring scenario: no more cores on the chip, but investing on the accelerators on the die (reduce movement of the data)
    - Intel Sandy Bridge has an integrated GPU
    - AMD has Fusion
    - NVIDIA is working with ARM
  - This open to heterogeneous cores
- HPC rule: whatever will be better to use we will use it
  - Exascale systems expected by 2018-2020
    - A factor 100x in performance, with the same power consumption as it is now!
- However we are not HPC! We don't need FLOPs in few kernels (e.g. Algebra)
  - Our application can be considered "real-life" applications
- In any case heterogeneous systems will be there
  - Cores will be not an issue! Entering in "Cores for free"-era

We are not alone

- Several projects on how to parallelize “real” applications on heterogeneous systems
  - Provide tools for auto-tuning, checking and profiling
  - High level and not invasive changes in the code, e.g. using directives
  - Incremental parallelism
- All vendors are moving in this direction:
  - Nobody thinks that a world of MPI+OpenMP+CUDA (which is the common situation in HPC) will be affordable in the long run even for HPC!
  - Intel is working on a very elegant integration of accelerators
    - MIC is a x86-64 compatible
  - PGI, Cray, CAPS and NVIDIA are working on a new product OpenACC
  - OpenMP 4.0
- It is reasonable to think that some techniques used now will become low-level in the next future
  - Like programming in assembly nowadays
  - Compiler and JIT tools

http://paraphrase-ict.eu/



You are here: Home

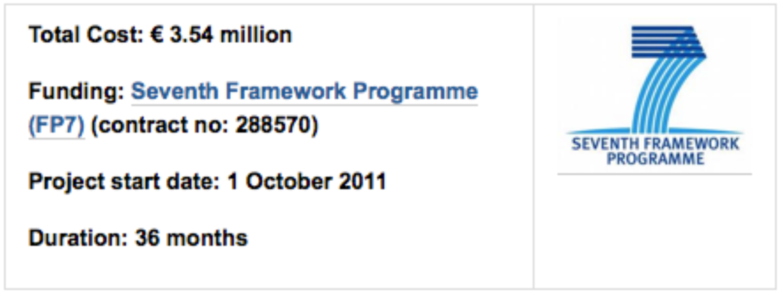
ParaPhrase

Parallel Patterns for Adaptive Heterogeneous Multicore Systems

The ParaPhrase project aims to produce a new structured design and implementation process for heterogeneous parallel architectures, where developers exploit a variety of parallel patterns to develop component based applications that can be mapped to the available hardware resources, and which may then be dynamically re-mapped to meet application needs and hardware availability.

Key Features

- Sustainable parallel computing through enhanced programmability and lower power consumption.
- Cost reduction in programmability and implementation of parallel systems.
- Better resource utilisation of parallel heterogeneous CPU/GPU architectures.



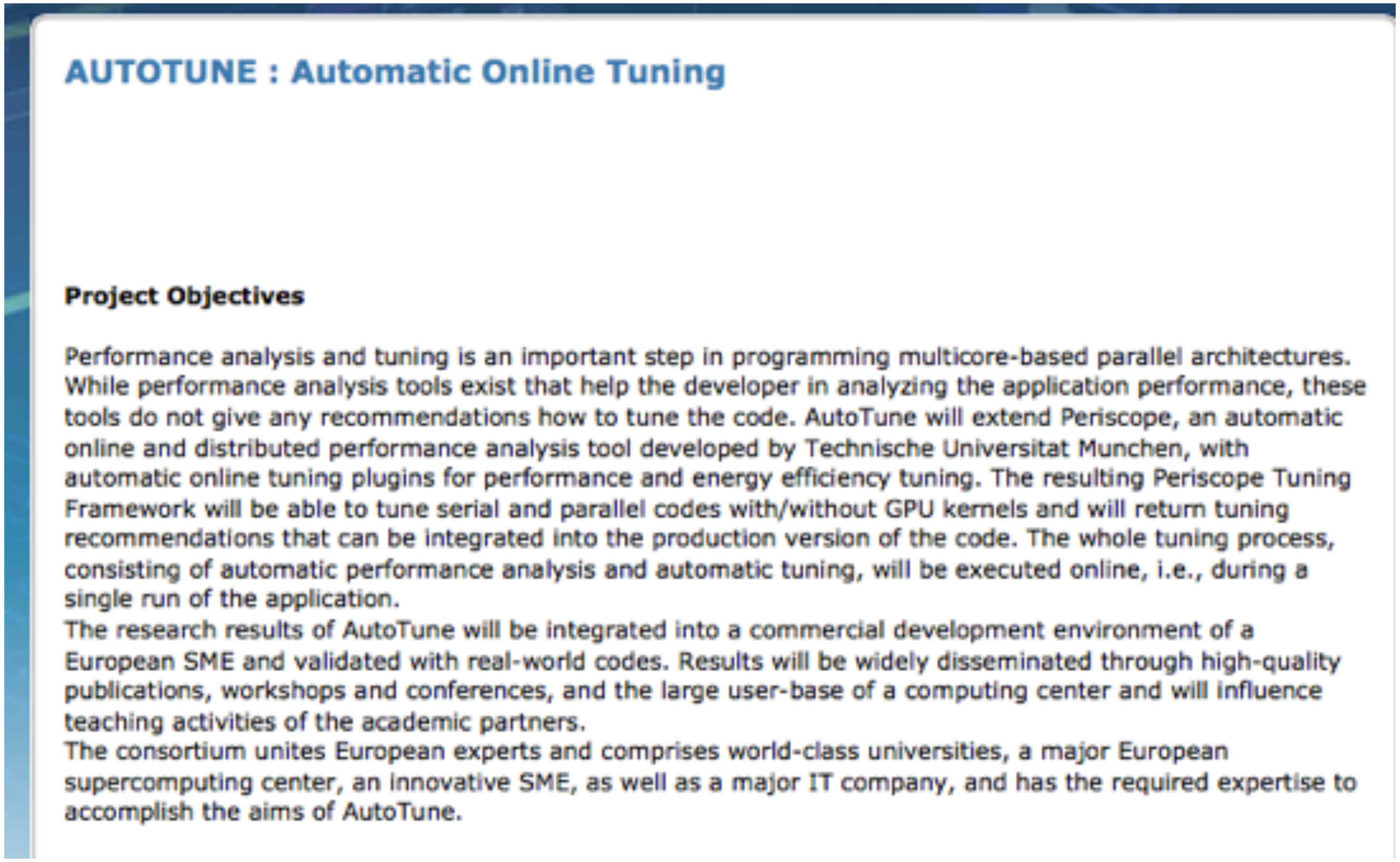
Total Cost: € 3.54 million

Funding: Seventh Framework Programme (FP7) **(contract no: 288570)**

Project start date: 1 October 2011

Duration: 36 months

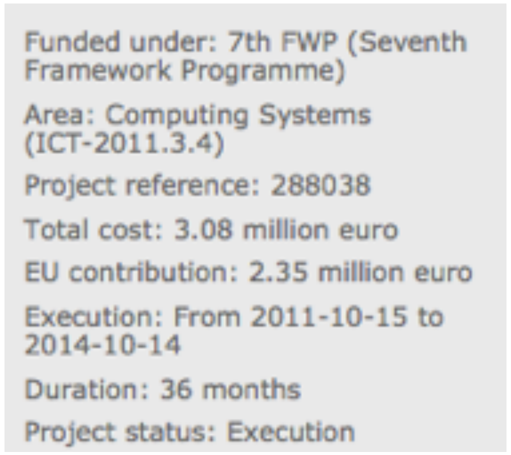
http://cordis.europa.eu/fetch?CALLER=PROJ_ICT&ACTION=D&CAT=PROJ&RCN=99932



AUTOTUNE : Automatic Online Tuning

Project Objectives

Performance analysis and tuning is an important step in programming multicore-based parallel architectures. While performance analysis tools exist that help the developer in analyzing the application performance, these tools do not give any recommendations how to tune the code. AutoTune will extend Periscope, an automatic online and distributed performance analysis tool developed by Technische Universitat Munchen, with automatic online tuning plugins for performance and energy efficiency tuning. The resulting Periscope Tuning Framework will be able to tune serial and parallel codes with/without GPU kernels and will return tuning recommendations that can be integrated into the production version of the code. The whole tuning process, consisting of automatic performance analysis and automatic tuning, will be executed online, i.e., during a single run of the application.
The research results of AutoTune will be integrated into a commercial development environment of a European SME and validated with real-world codes. Results will be widely disseminated through high-quality publications, workshops and conferences, and the large user-base of a computing center and will influence teaching activities of the academic partners.
The consortium unites European experts and comprises world-class universities, a major European supercomputing center, an innovative SME, as well as a major IT company, and has the required expertise to accomplish the aims of AutoTune.



Funded under: 7th FWP (Seventh Framework Programme)

Area: Computing Systems (ICT-2011.3.4)

Project reference: 288038

Total cost: 3.08 million euro

EU contribution: 2.35 million euro

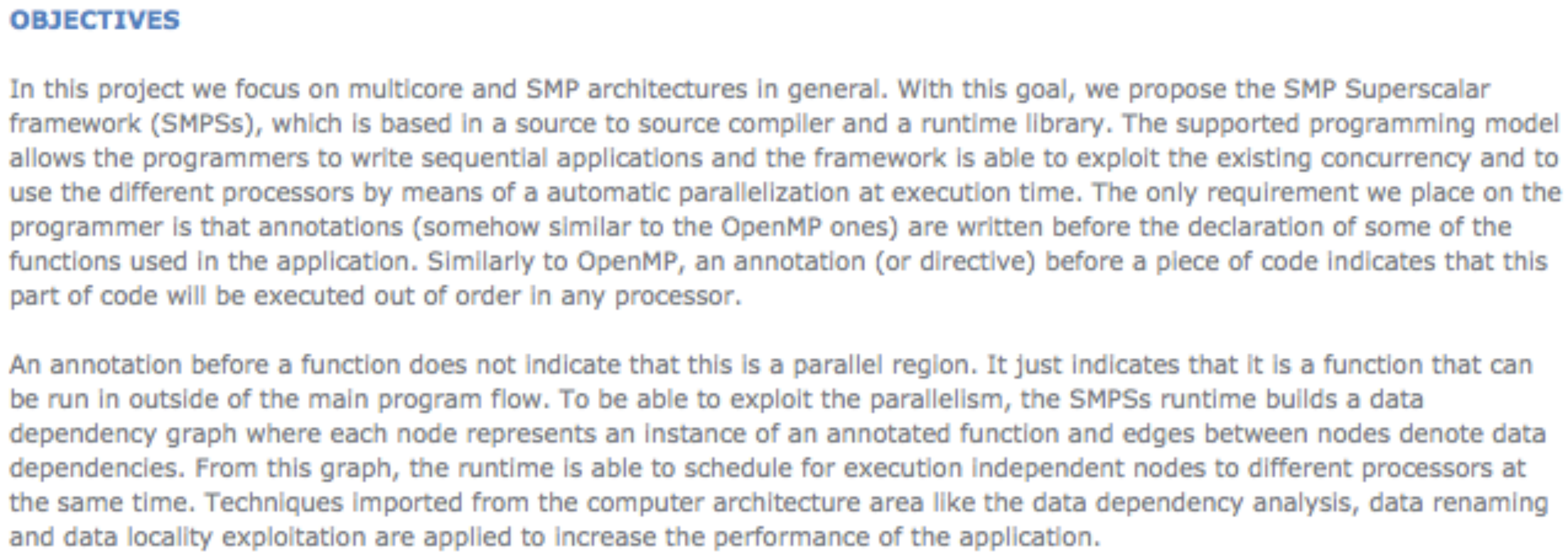
Execution: From 2011-10-15 to 2014-10-14

Duration: 36 months

Project status: Execution

http://www.bsc.es/plantillaG.php?cat_id=385

SMP superscalar
Spain projects (BCS)



OBJECTIVES

In this project we focus on multicore and SMP architectures in general. With this goal, we propose the SMP Superscalar framework (SMPSs), which is based in a source to source compiler and a runtime library. The supported programming model allows the programmers to write sequential applications and the framework is able to exploit the existing concurrency and to use the different processors by means of a automatic parallelization at execution time. The only requirement we place on the programmer is that annotations (somehow similar to the OpenMP ones) are written before the declaration of some of the functions used in the application. Similarly to OpenMP, an annotation (or directive) before a piece of code indicates that this part of code will be executed out of order in any processor.

An annotation before a function does not indicate that this is a parallel region. It just indicates that it is a function that can be run in outside of the main program flow. To be able to exploit the parallelism, the SMPSs runtime builds a data dependency graph where each node represents an instance of an annotated function and edges between nodes denote data dependencies. From this graph, the runtime is able to schedule for execution independent nodes to different processors at the same time. Techniques imported from the computer architecture area like the data dependency analysis, data renaming and data locality exploitation are applied to increase the performance of the application.

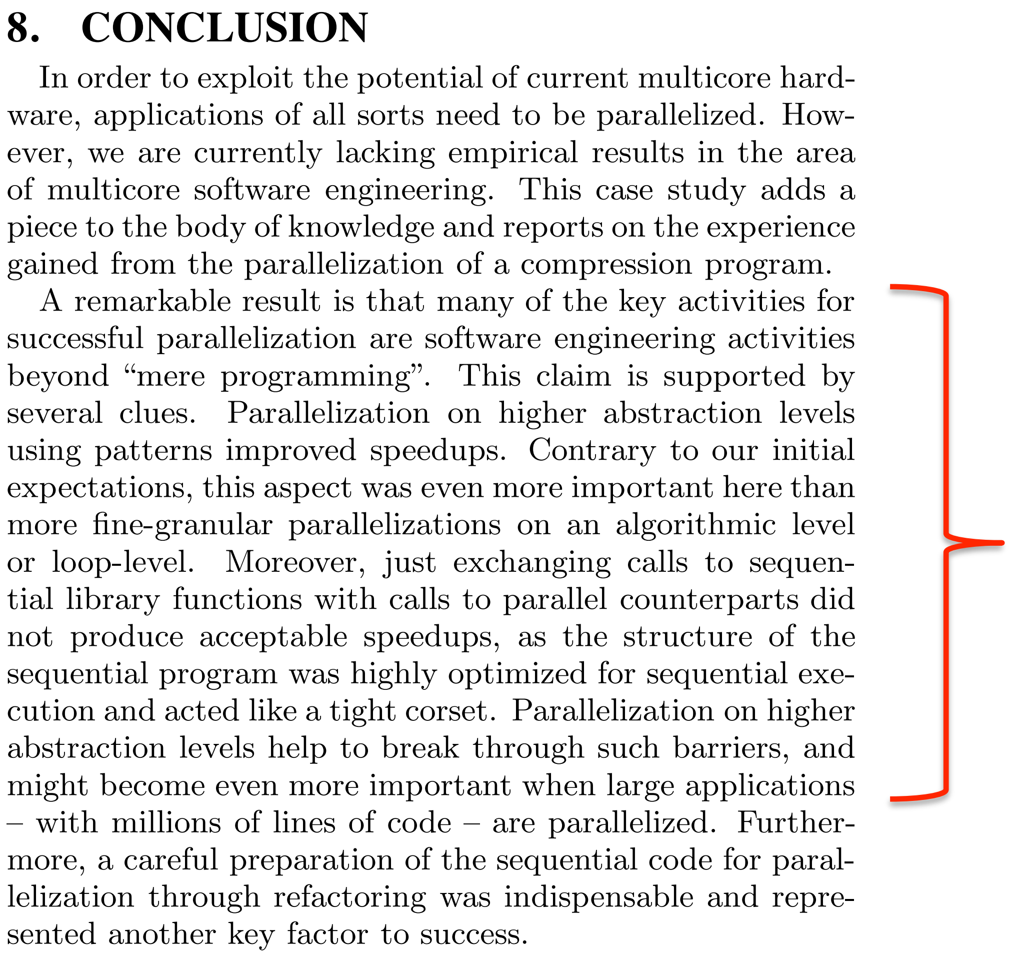
Other links

- Interesting reading by Victor Pankratius, KIT, German (http://www.victorpankratius.com/)
  - http://www.rz.uni-karlsruhe.de/~kb95/papers/Pankratius-SoftwareEngineeringInTheEraOfParallelism.pdf

Parallelizing BZip2

A Case Study in Multicore Software Engineering

Victor Pankratius, Ali Jannesari, Walter F. Tichy. *IEEE Software 26(6), pp. 70-77, Nov.-Dec. 2009, ISSN: 0740-7459, DOI: 10.1109/MS.2009.183*



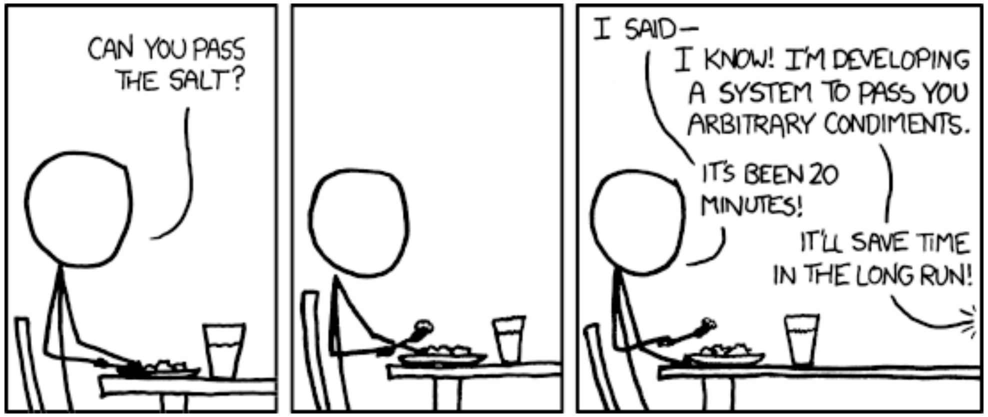
8. CONCLUSION

In order to exploit the potential of current multicore hardware, applications of all sorts need to be parallelized. However, we are currently lacking empirical results in the area of multicore software engineering. This case study adds a piece to the body of knowledge and reports on the experience gained from the parallelization of a compression program.

A remarkable result is that many of the key activities for successful parallelization are software engineering activities beyond "mere programming". This claim is supported by several clues. Parallelization on higher abstraction levels using patterns improved speedups. Contrary to our initial expectations, this aspect was even more important here than more fine-granular parallelizations on an algorithmic level or loop-level. Moreover, just exchanging calls to sequential library functions with calls to parallel counterparts did not produce acceptable speedups, as the structure of the sequential program was highly optimized for sequential execution and acted like a tight corset. Parallelization on higher abstraction levels help to break through such barriers, and might become even more important when large applications – with millions of lines of code – are parallelized. Furthermore, a careful preparation of the sequential code for parallelization through refactoring was indispensable and represented another key factor to success.

How we can proceed?

- Look for high level parallelism (a la Intel CnC)
  - Processes and tasks
  - Thread-safety
- Work on tools
  - Apply incremental parallelism
- Collaboration with other projects
  - We need to think in the “future”
  - Develop tools to check correctness or auto-parallelization
- Extract some part of the code and work on it
  - At the beginning we should not consider optimization and efficiency
  - It can be that parallelization will introduce overhead, but this is the winning solution in the long run (cores are for free)
- Accelerators can play a role in our code only in the second phase
- Need to focus on specific problems
  - Debugging
  - Correctness
  - Specific algorithms (log, I/O, loop parallelism, task parallelism, random numbers, data structures…)


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!
Credits to xkcd

How we can proceed?

- The current situations seems to require the development of two versions of the code
  - Sequential for production and checking
  - Parallel for developments
  - Can we keep this situation in the long run?
    - The two versions can diverge at some point, so it would require doubling the work
- Moreover: keep the physics involved!
  - I think this is the critical point, unless everything works out of the box in a parallel world (at the moment we cannot guaranteed that in a much easier sequential world)

Conclusion

- My hope is that the answer to the initial question "Why parallelize?" will be "For doing more work"
  - I don't think it is easy to convince someone that we need to parallelize the code as it is now, since it is working!
    - We should clarify the goal in the long run, i.e. what the experiments want
  - Which new workload can be added by the experiments in the long run?
    - Better tracking, more complex algorithms, pileup...?
    - Increase collisions rate by LHC? (see HiLumi project, http://indico.cern.ch/conferenceDisplay.py?confId=150474)