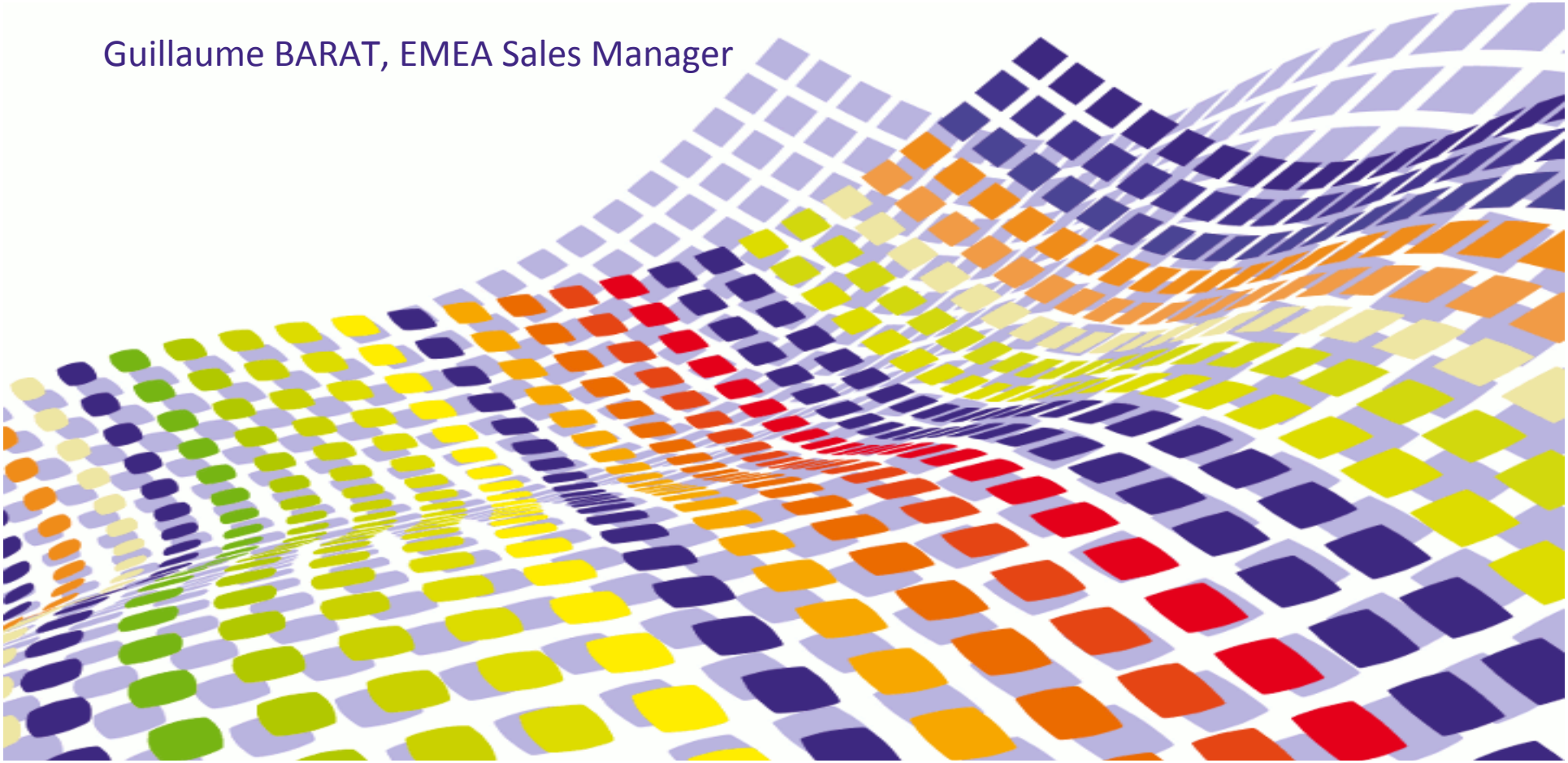
CAPS

How to Write Code that Will Survive the Many-Core Revolution

Write Once, Deploy Many(-Cores)

Guillaume BARAT, EMEA Sales Manager



CAPS worldwide ecosystem

CAPS

Customers

AWE, BNP PARIBAS, cea, GENCI, EADS, DENSO Member of the TOYOTA Group, TOTAL, Statoil, OAK RIDGE National Laboratory, NOAA, HUAWEI, HLRS, 東北大学 TOHOKU UNIVERSITY, ICHEC, ifp

Business Partners

allinea DDT, absoft, AMD Smarter Choice, intel, hp, IBM, CULA, ACCELEREYES, NVIDIA, ARM, BULL, NEC, sgi

- Involved in many European Research projects

PRACE

- OpenMP ARB Accelerator program subcommittee

OpenMP

- OpenStandard Initiative

open hmpp

OpenACC DIRECTIVES FOR ACCELERATORS

- HMPP Competences Centers in Europe and Asia

hmpp Competence Center

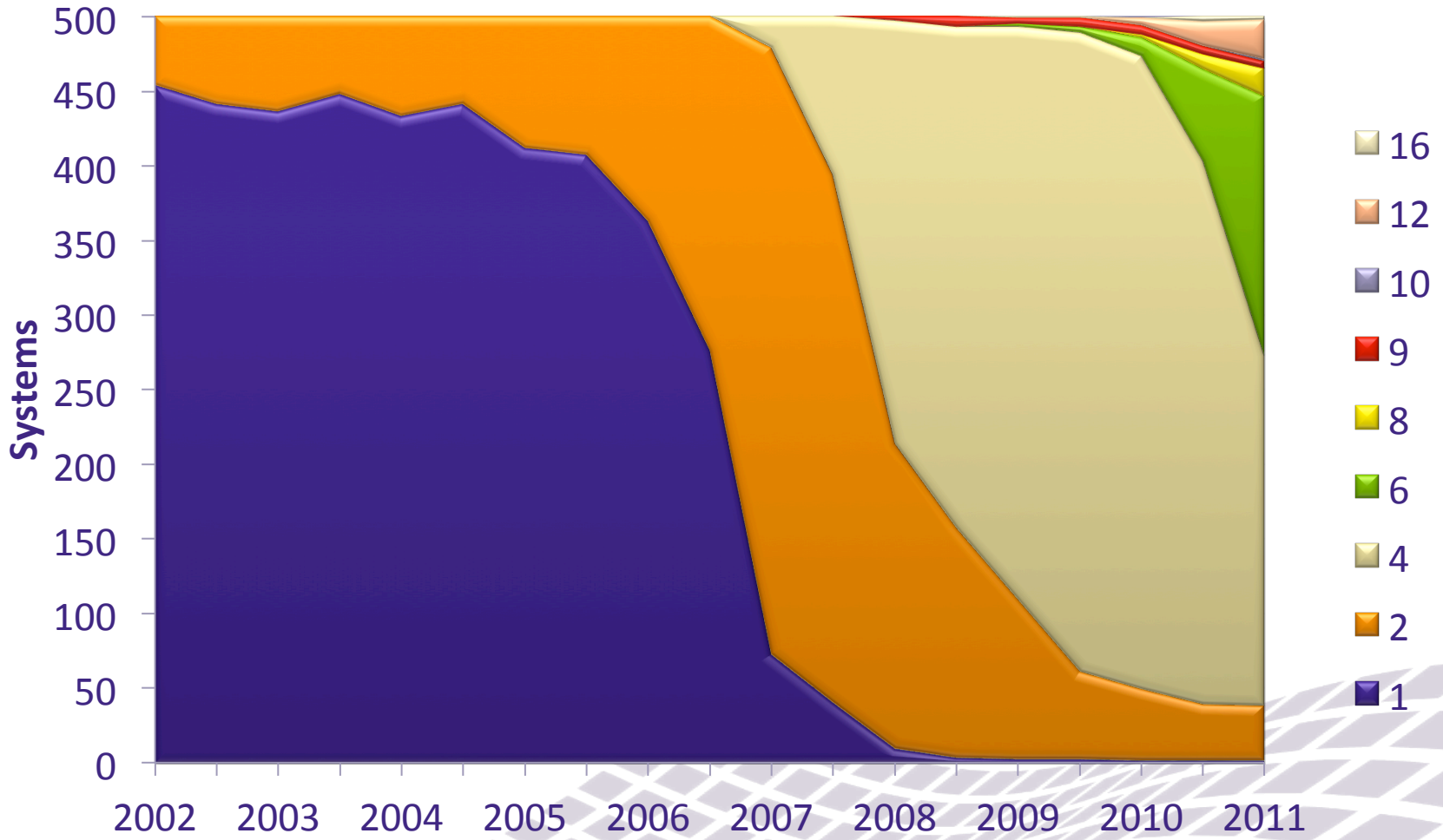
Foreword

- "How to write code that will survive the many-core revolution?" is being setup as a collective initiative of HMPP Competence Centers
  - Gather competences from universities across the world (Asia, Europe, USA) and R&D projects: H4H, Autotune, TeraFlux
  - Visit http://competencecenter.hmpp.org for more information

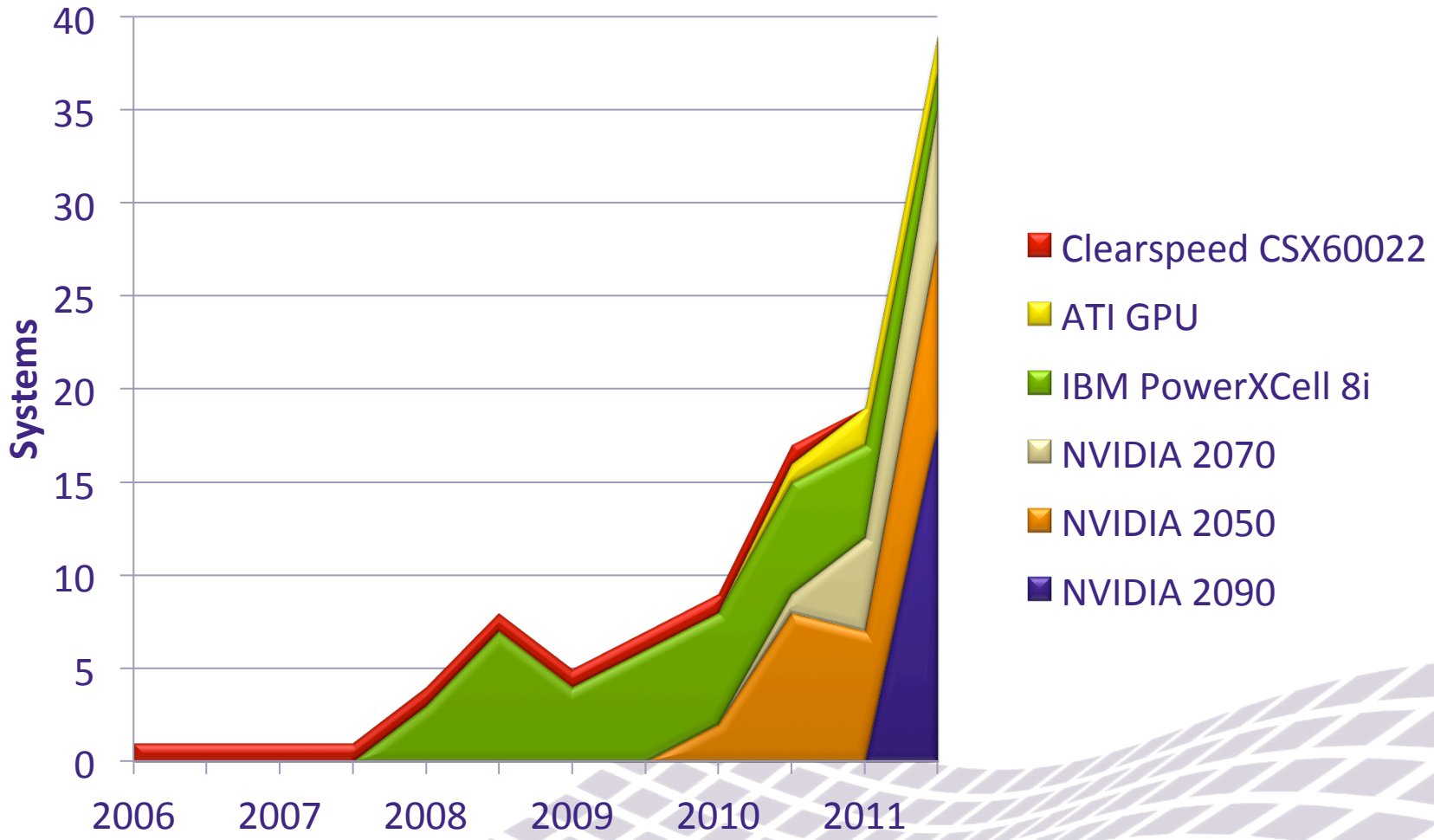
UNIVERSITY OF HOUSTON

COMPUTER ARCHITECTURE GROUP
UNIVERSITY OF A CORUÑA
GAC.UDC.ES

Trend from Top500: Cores per Socket



Trend from Top500: Accelerators



IDC @SC11: Study Results

CAPS

- Many HPC codes aren't seeing a speed up with new hardware systems (due to many-core, lower bandwidth, lower memory/core, etc.)
  - Many applications will need a major redesign
  - Multi-core will cause many issues to "hit-the-wall"
  - So GPUs can offer a speed-up

- BUT
  - GPU still need to be more easy to program
  - Future portability is a key concern

SC11: Consensus about directives ?

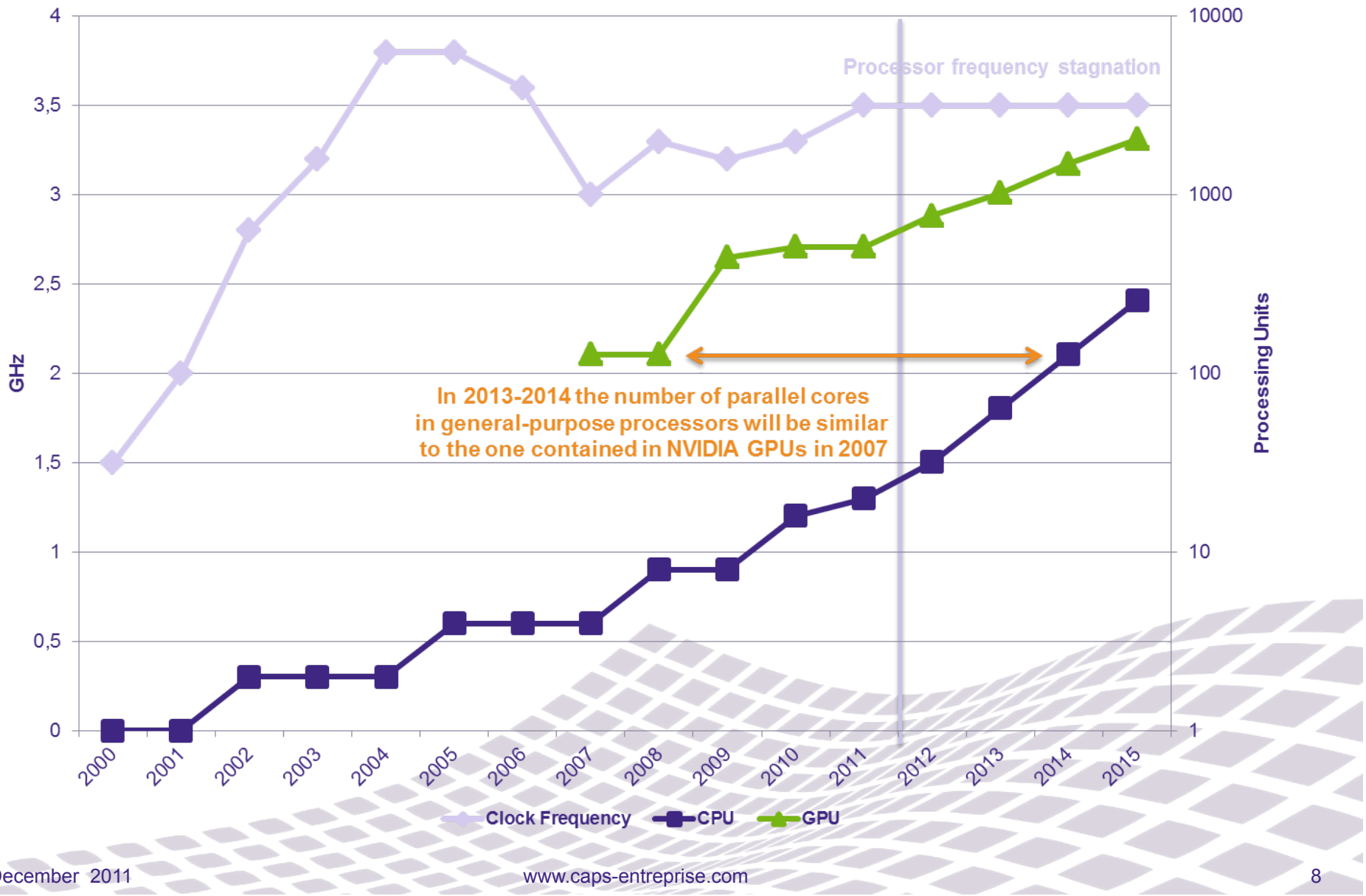
CAPS

- OpenACC initiative (2011)
  - CAPS, Cray, NVIDIA, PGI
  - A first common syntax for accelerator regions
  - Visit http://www.openacc-standard.com for more information

OpenACC
DIRECTIVES FOR ACCELERATORS

- OpenHMPP initiative (2010)
  - Directive Open Standard for many-core programming
  - Complient with OpenACC syntax
  - Topics not covered by OpenACC: *data-flow extension, tracing interface, auto-tuning APIs…*
  - Visit http://www.openhmpp.org for more information

Where Are We Going?



What to Expect From the Architecture?

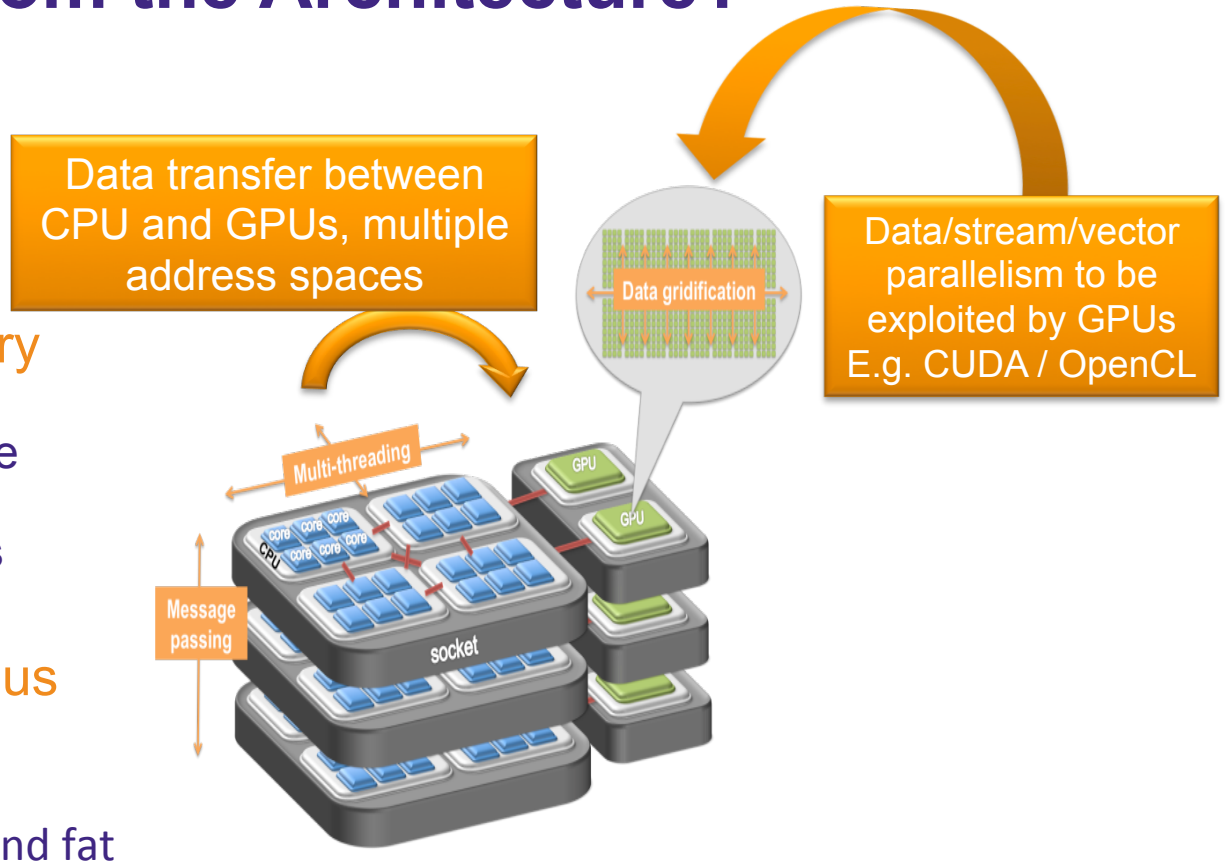
CAPS

Strong Non Uniform Memory Access (NUMA) effects

- Not all memories may be coherent
- Multiple address spaces

Node is not an homogeneous piece of hardware

- Not only one device
- Balance between small and fat cores changes over time



Many parallelism forms are needed to deal with

- Increasing number of processing units
- Some form of vector computing (AVX or SSE instructions)

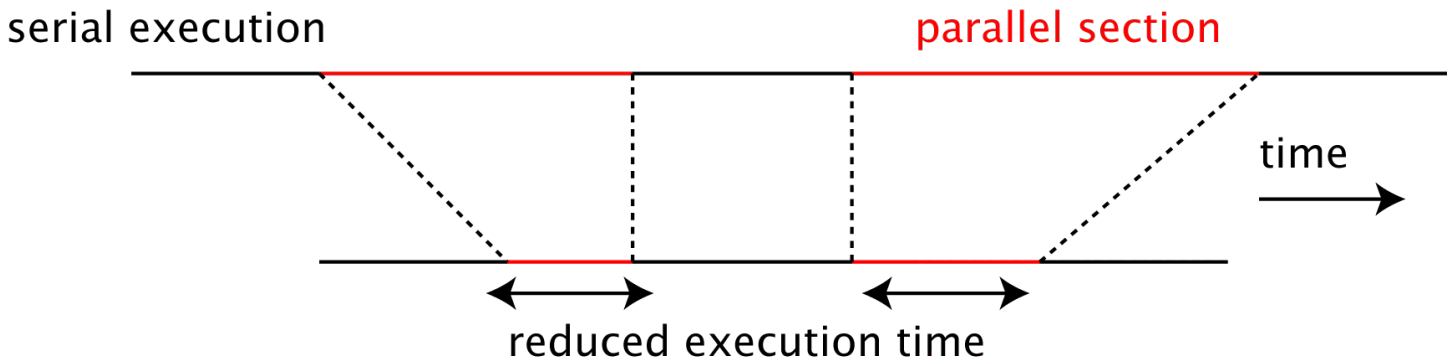
What to consider by writing your code

CAPS

- Computing power comes from parallelism
  - Hardware (frequency increase) to software (parallel codes) shift
  - Driven by energy consumption
    - Heterogeneity is the source of efficiency
    - Few large fast OO cores combined with many smaller cores (e.g. APUs)
- Fast moving hardware targets environment
  - e.g. fast GPU improvements (RT and HW), new massively parallel CPU
  - Write codes that will last many architecture generations
- Keeping a unique version of the codes, preferably mono-language, is a necessity
  - Reduce maintenance cost
  - Directive-based approaches suitable
  - Preserve code assets

Software Main Driving Forces

- ALF - Amdahl's Law is Forever
  - A high percentage of the execution time has to be parallel
  - Many algorithms/methods/techniques will have to be reviewed to scale



- Data locality is expected to be the main issue
  - Moving data will always suffer latency

One MPI Process per Core Approach

CAPS

- Benefit
  - No need to change existing code
- Issues
  - Extra latency compared to shared memory use
    - MPI implies some copying required by its semantics (even if efficient MPI implementations tend to reduce them)
  - Excessive memory utilization
    - Partitioning for separate address spaces requires replication of parts of the data.
    - When using domain decomposition, the sub-grid size may be so small that most points are replicated (i.e. ghost zone)
    - Memory replication implies more stress on the memory bandwidth which finally produces a weak scaling
  - Cache trashing between MPI processes
  - Heterogeneity management
    - How are MPI processes linked to accelerator resources?
    - How to deal with different core speed?
    - The balance may be hard to find between the optimal MPI process that makes good use of the CPU core and the use of the accelerators that may be more efficiently used with a different ratio

Thread Based Parallelism Approach

CAPS

- Benefits
  - Some codes are already ported to OpenMP/threads APIs
  - Known APIs

- Issues
  - Data locality and affinity management
    - Data locality and load balancing (main target of thread APIs) are in general two antagonistic objectives
    - Usual thread APIs make it difficult / not direct to express data locality affinity
  - Reaching a tradeoff between vector parallelism (e.g. using the AVX instruction set), thread parallelism and MPI parallelism
    - Vector parallelism is expected to impact more and more on the performance
    - Current threads APIs have not been designed to simplify the implementation of such tradeoff
  - Threads granularity has to be tuned depending on core characteristics (e.g. SMT, heterogeneity)
    - Thread code writing style does not make it easy to tune

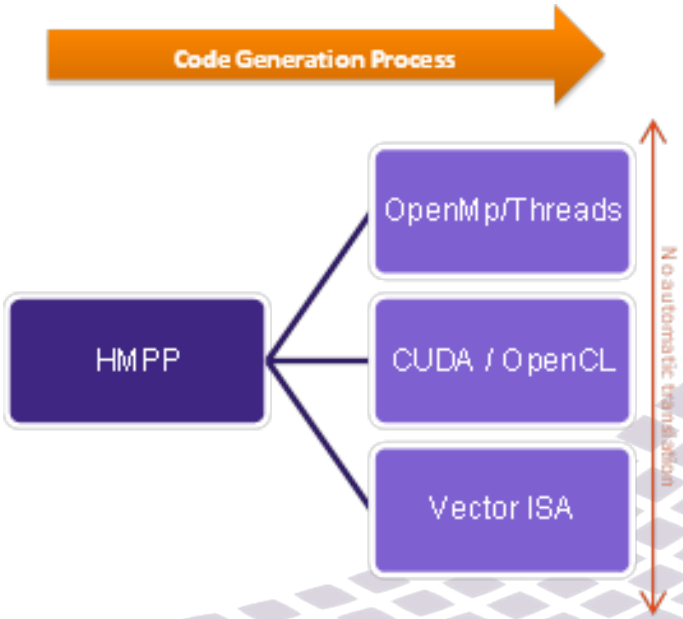
An Approach for Portable Many-Core Codes

CAPS

1. Try to expose **node** massive data parallelism in a target independent way
2. Do not hide parallelism by awkward coding
3. Keep code debug-able
4. Track performance issues
5. Exploit libraries
6. When allowed, do not target 100% of possible performance, the last 20% are usually very intrusive in the code

1 - Express Parallelism, not Implementation

- Rely on code generation for implementation details
  - Usually not easy to go from a low level API to another low level one
  - Tuning has to be possible from the high level
  - But avoid relying on compiler advanced techniques for parallelism discovery, …
  - **You may have to change the algorithm**!
- An example with HMPP



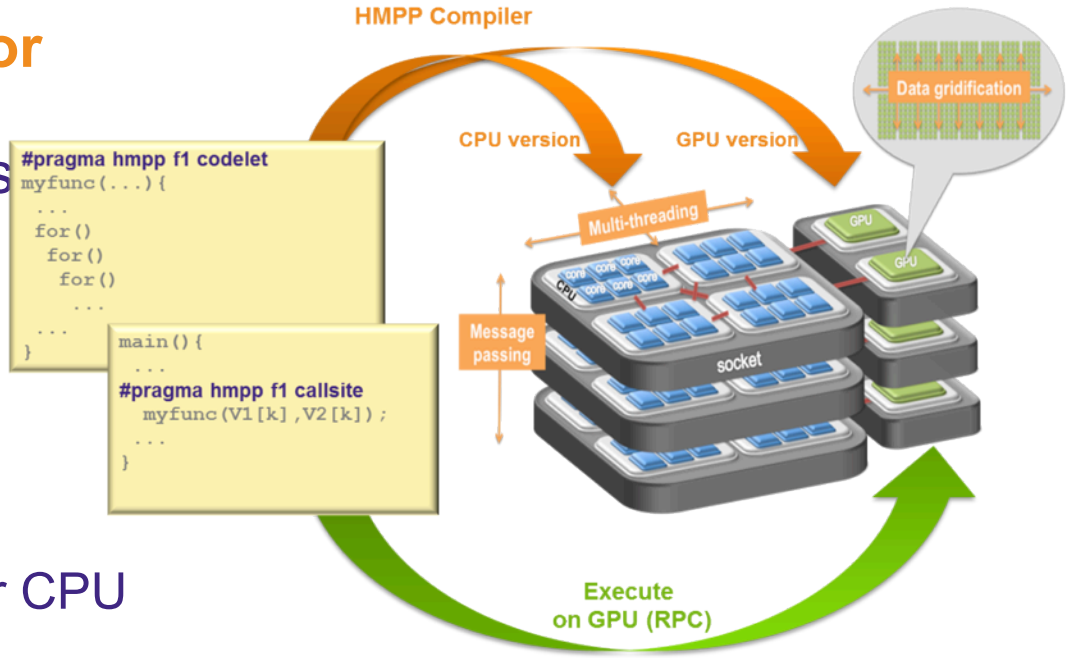

```
#pragma hmppcg gridify(j,i)
#pragma hmppcg unroll(4), jam(2)
  for( j = 0 ; j < p ; j++ ) {
    for( i = 0 ; i < m ; i++ ) {
     for (k = ...) { ...}
      vout[j][i] = alpha * ...;
    }
  }
```

HMPP Hybrid Compiler

CAPS

Directive based approach for many-core

Cuda & **OpenCL** devices
and soon **Intel MIC**



- Features
  - With **one source code**, target **multiple many-core architectures**
  - Distribute computation over CPU & GPU cores (**Multi-GPU**)
  - High many-core performance with optimized **data management**
  - **Libraries interoperability** with user code
  - Protect your software investment by using an **Open Standard**

Rapidly develop Many-core
Accelerated applications

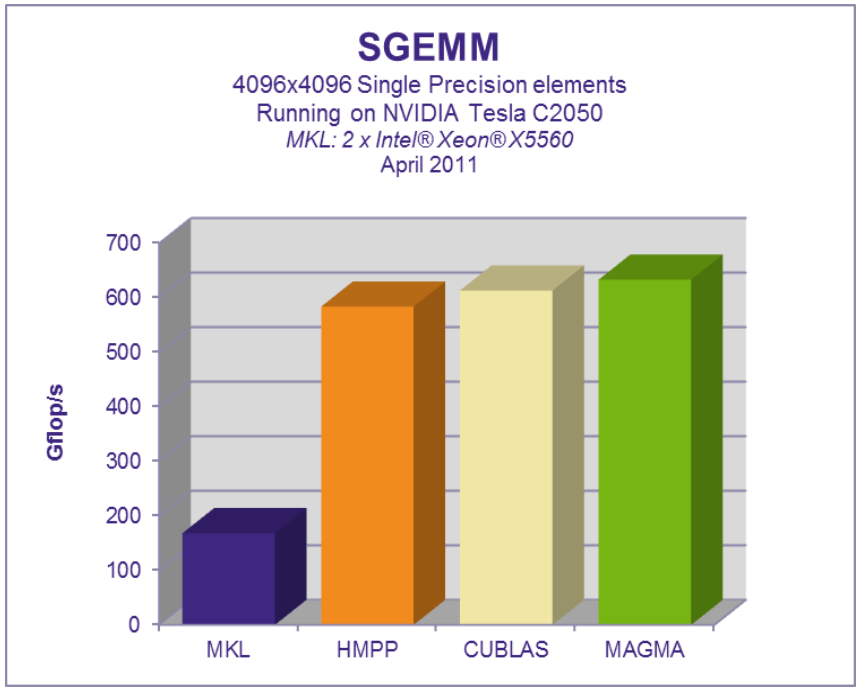
Rich set of directives for performance

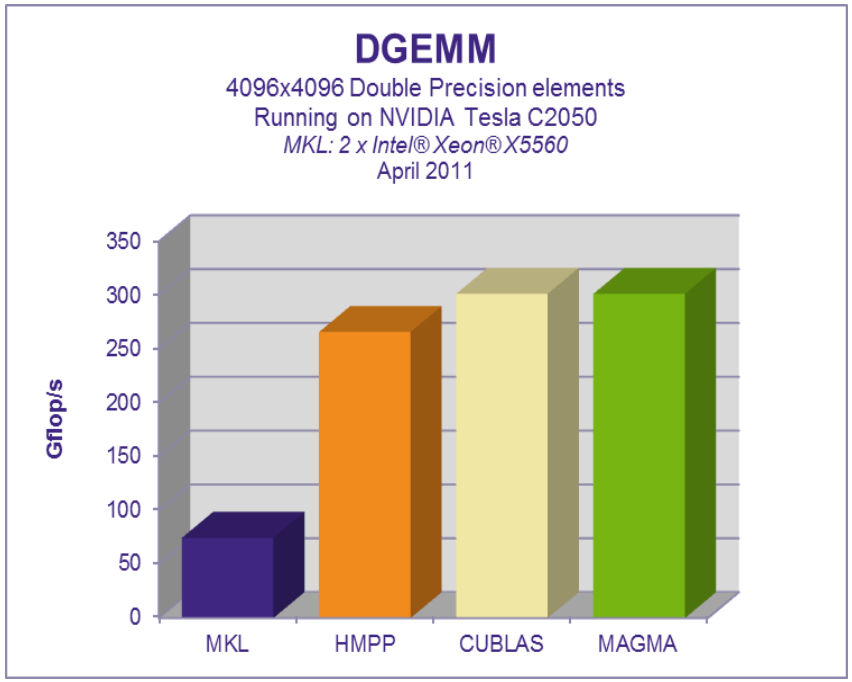
CAPS

- Manycore programming directives in legacy code
  - Declare and generate GPU versions of computations (*codelets)*
  - Optimize data movement
  - Distribute computations over CPU cores & GPUs
- Tuning of GPU kernels
  - Advanced code optimizations
  - Control mapping of computations
  - Fully exploit GPU stream architecture

Performance does matter

En average, HMPP reaches Cuda performance +/- 10%





2 x Intel(R) Xeon(R) X5560 @ 2.80GHz (8 cores) - MKL
NVidia Tesla C2050, ECC activated – HMPP, CUBLAS, MAGMA

energie atomique • energies alternatives

Performances (max)
NO TRANSFERTS

T in s

Scalar	18457.52	3030.9	4668.67	7531.1	1054.4	15.68	1933.82
OMP=8	5040.1	379.09	1479.91	1302.64	1248.35	6.76	348.32
HMPP	820.34	56.88	388.23	156.61	81.58	2.04	29.7
CUDA	1267.66	75.01	458.95	135.61	66.99	66.51	531.3
	ALL	NOISE	DIFFUS	KERSBS	SHIFT	BOUND	FLUX
OMP / SEQ	3.66	8.00	3.15	5.78	0.84	2.32	5.55
HMPP / SEQ	22.50	53.29	12.03	48.09	12.92	7.69	65.11
CUDA / SEQ	14.56	40.41	10.17	55.53	15.74	0.24	3.64

Speedup

Diffus = FFT FW + diffrac + FFTBW
KERSBS = KER + SBS

Geom: 128 x 128 x 256
No I/O
1 MPI

2 – Do not hide parallelism

CAPS

- Do not hide parallelism by awkward coding
  - Data structure aliasing, …
  - Deep routine calling sequences
  - Separate concerns (functionality coding versus performance coding)

- Data parallelism when possible
  - Simple form of parallelism, easy to manage
  - Favor data locality
  - But sometimes too static

- Kernels level
  - Expose massive parallelism
  - Ensure that data affinity can be controlled
  - Make sure it is easy to tune the ratio vector / thread parallelism

Data Structure Management

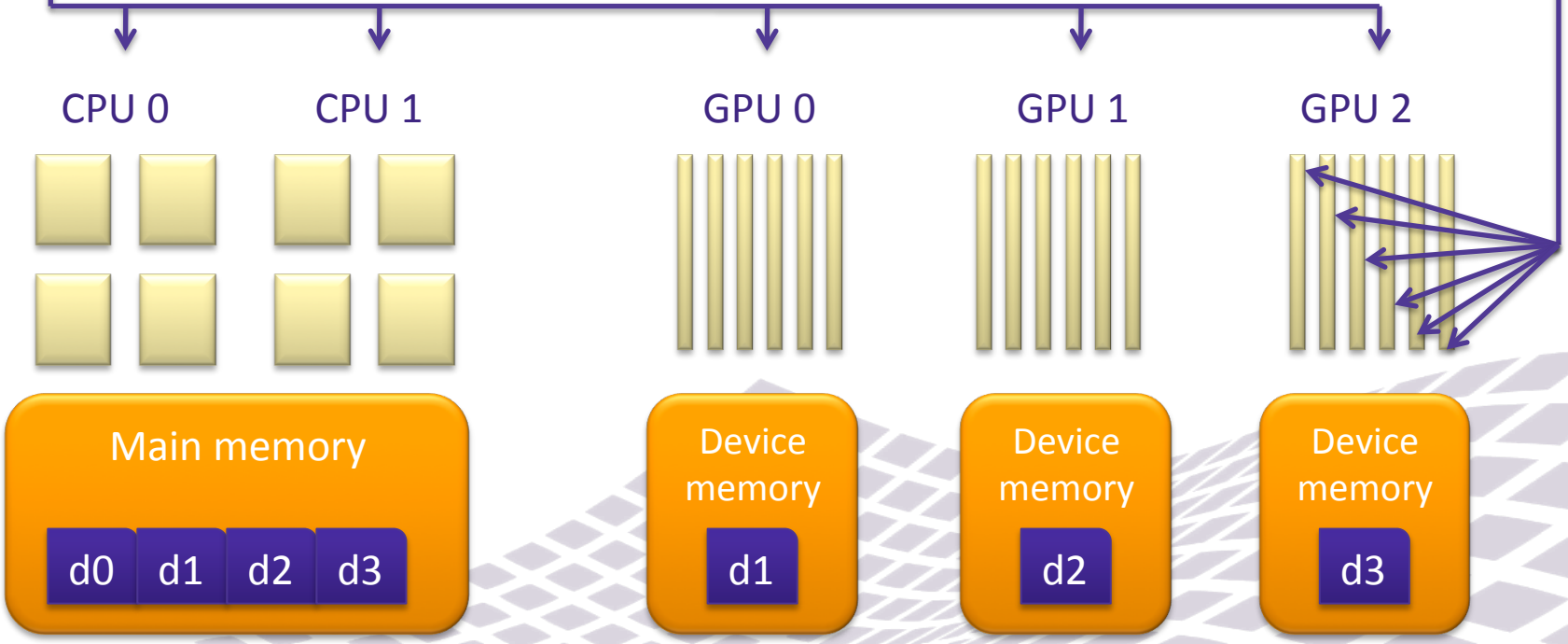
CAPS

- Data locality
  - Makes it easy to move from one address space to another one
  - Makes it easy to keep coherent
- Do not waste memory
  - Memory per core ratio not improving
- Choose simple data structures
  - Enable vector/SIMD computing
  - Use library friendly data structures
  - May come in multiple forms, e.g. sparse matrix representation
- For instance consider “data collections” to deal with multiple address spaces or multiple devices or parts of a device
  - Gives a level a adaptation for dealing with heterogeneity
  - Load distribution over the different devices is simple to express

HMPP 3.0 Map Operation on Data Collection

CAPS

```
#pragma hmpp <mgrp> parallel
for(k=0;k<n;k++) {
     #pragma hmpp <mgrp> f1 callsite
         myparallelfunc(d[k],n);
   }
```



3 - Debugging Issues

CAPS

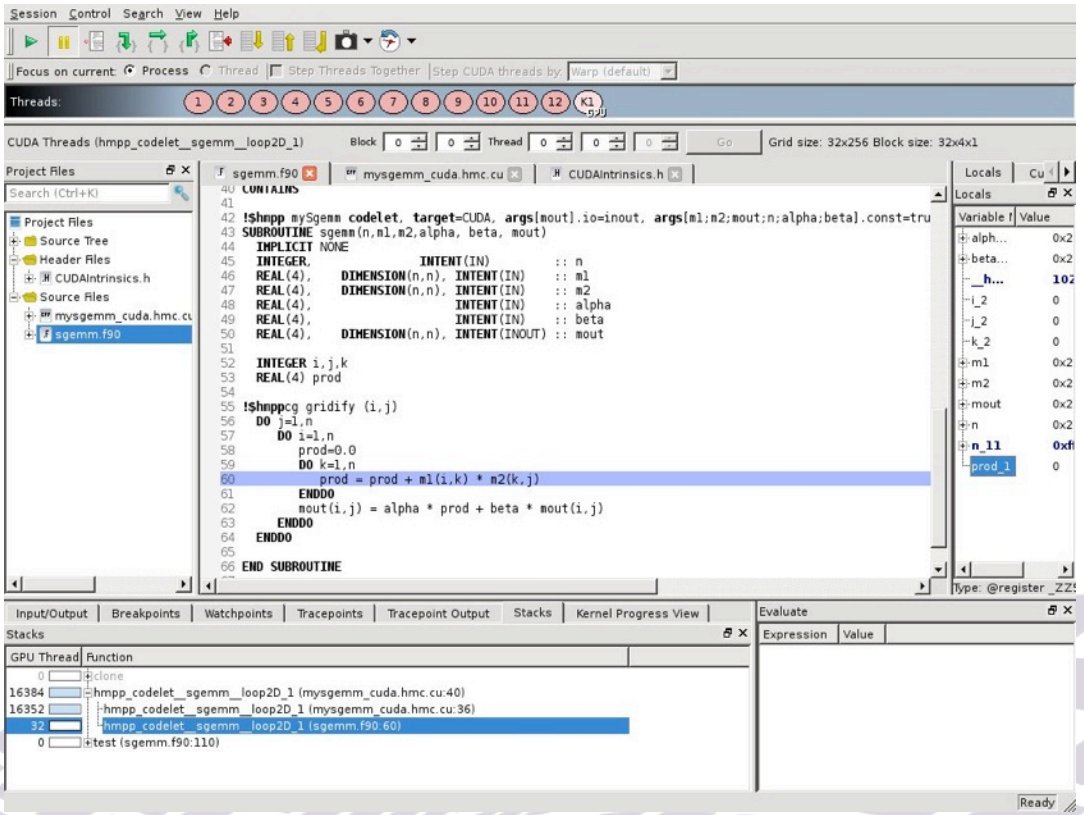
- Avoid assuming you won't have to debug!
- Keep serial semantic
  - For instance, implies keeping serial libraries in the application code
  - Directives based programming makes this easy
- Ensure validation is possible even with rounding errors
  - Reductions, …
  - Aggressive compiler optimizations
- Use defensive coding practices
  - Events logging, parameterize parallelism, add synchronization points, …
  - Use debuggers (e.g. Allinea DDT)

Allinea DDT – Debugging Tool for CPU/GPU

CAPS

- Debug your kernels:
  - Debug CPU and GPU concurrently
- Examine thread data
  - Display variables
- Integrated with HMPP
  - Allows HMPP directives breakpoints
  - Step into HMPP codelets

allinea ddt
the distributed debugging tool

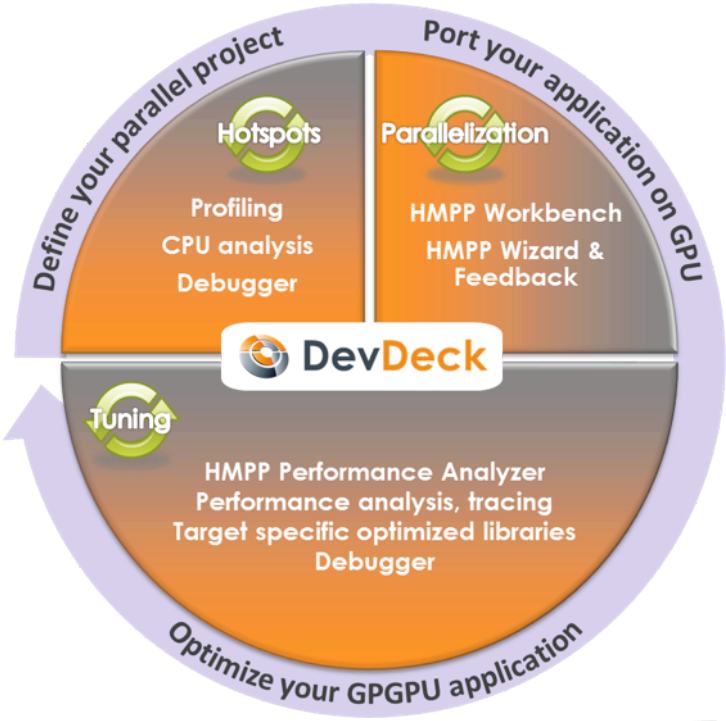


4 - Performance Measurements

- Parallelism is about performance!

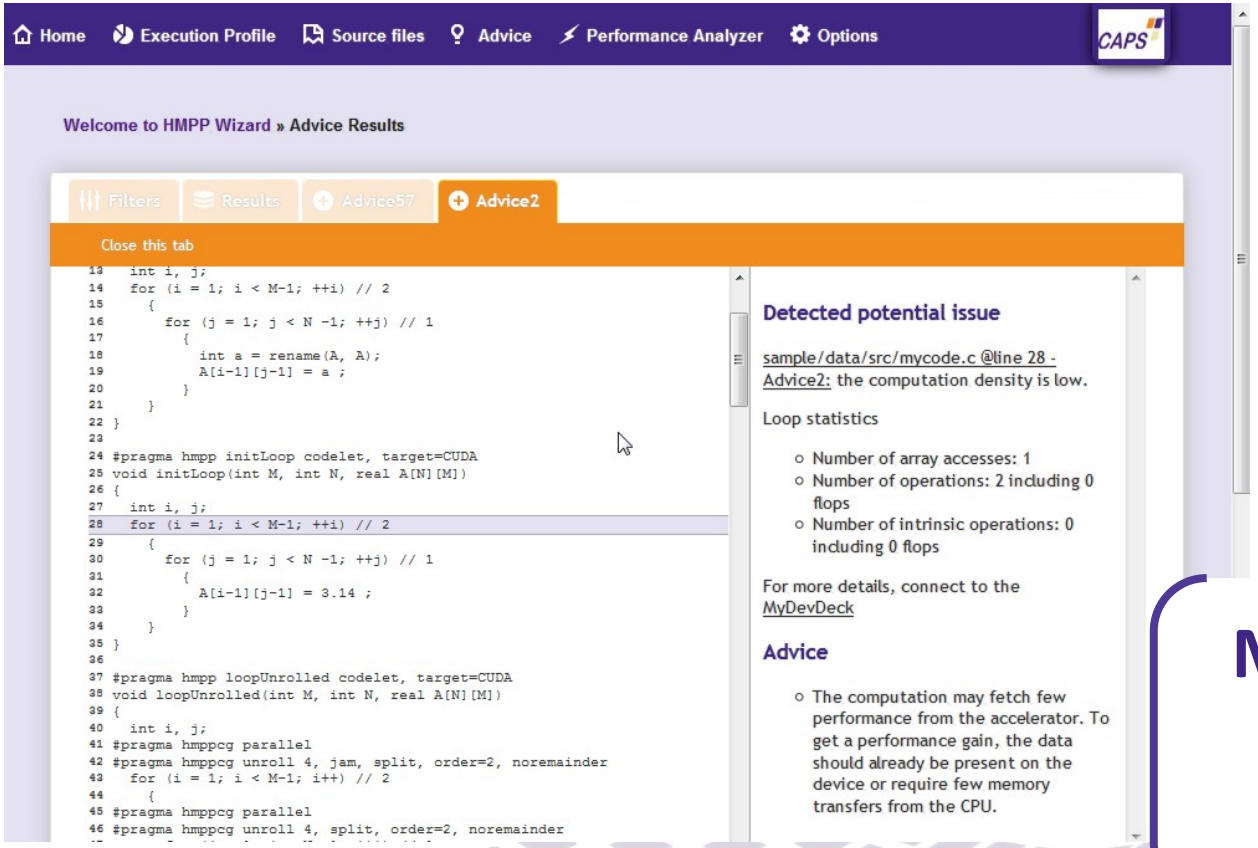
- Track Amdahl's Law Issues
  - Serial execution is a killer
  - Check scalability
  - Use performance tools

- Add performance measurement in the code
  - Detect bottleneck a.s.a.p.
  - Make it part of the validation process



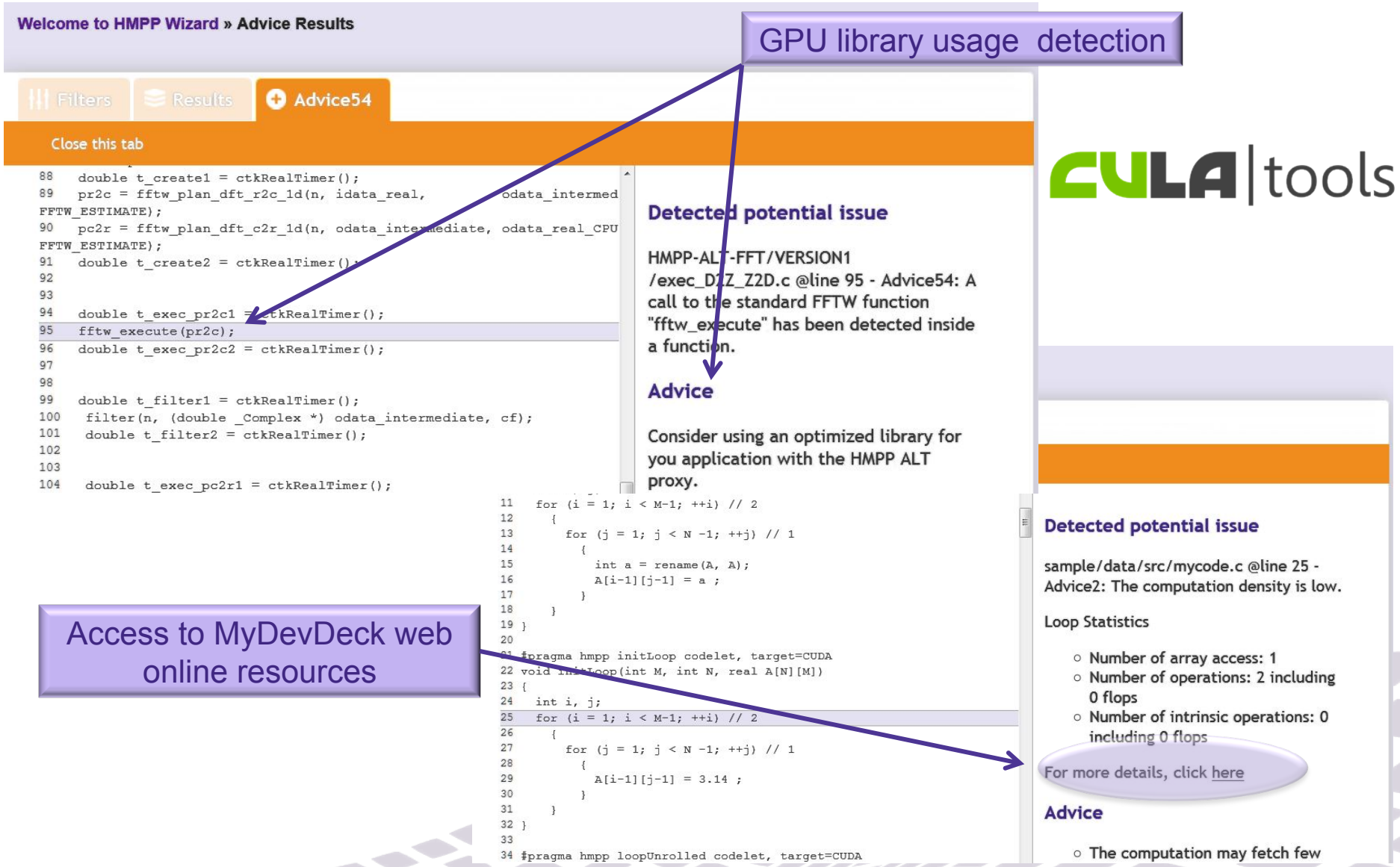
HMPP Wizard

- Analyze the code to help migrate to **many-core architectures** within an incremental process
- Provide **tuning advices**
  - Check coalescing of memory accesses
  - Improve parallelism …



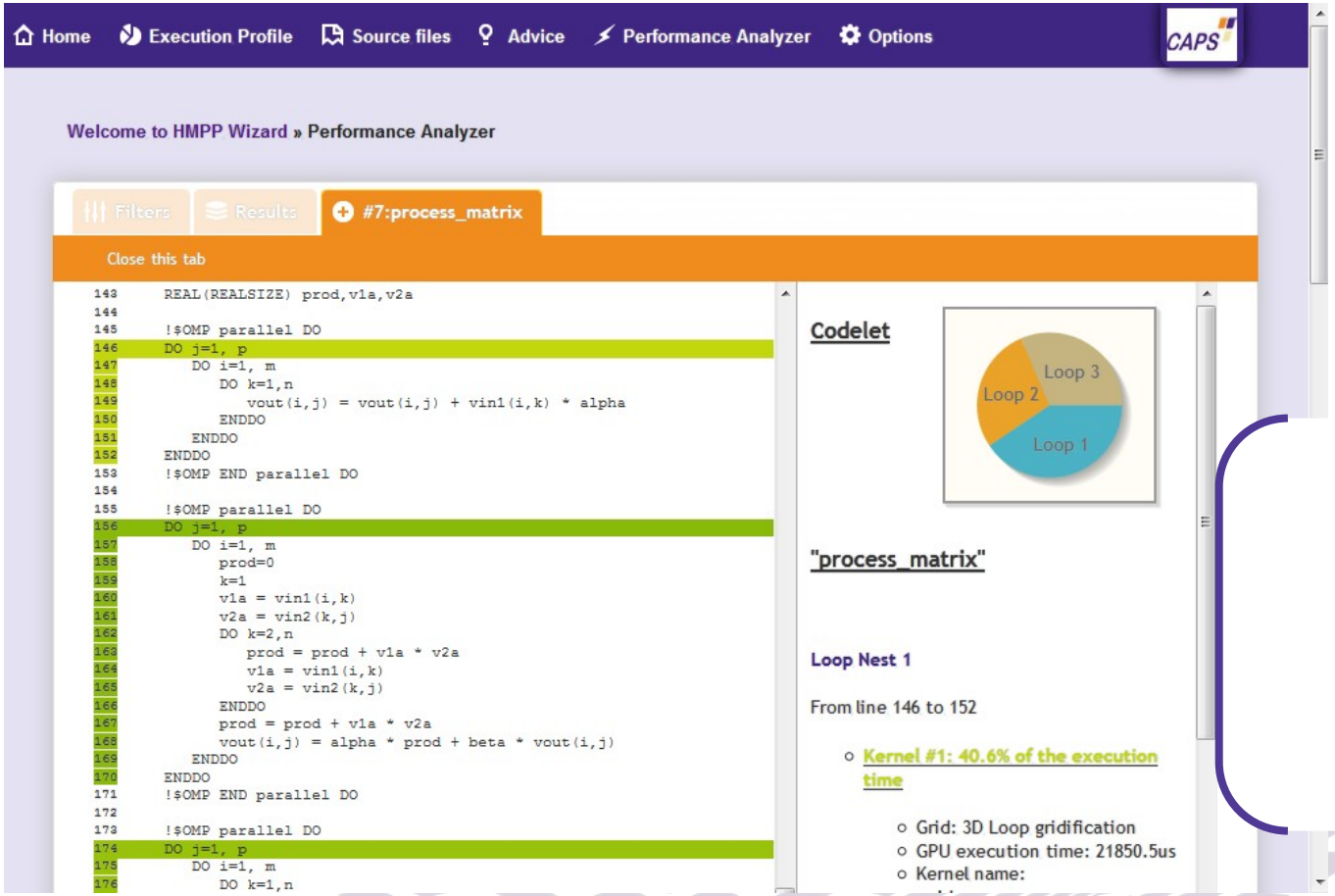
Make your Code Many-Core Friendly

HMPP Wizard


Welcome to HMPP Wizard » Advice Results
GPU library usage detection
Filters
Results
Advice54
Close this tab
88 double t_create1 = ctkRealTimer();
89 pr2c = fftw_plan_dft_r2c_1d(n, idata_real, odata_intermed
FFTW_ESTIMATE);
90 pc2r = fftw_plan_dft_c2r_1d(n, odata_intermediate, odata_real_CPU
FFTW_ESTIMATE);
91 double t_create2 = ctkRealTimer();
92
93
94 double t_exec_pr2c1 = ctkRealTimer();
95 fftw_execute(pr2c);
96 double t_exec_pr2c2 = ctkRealTimer();
97
98
99 double t_filter1 = ctkRealTimer();
100 filter(n, (double _Complex *) odata_intermediate, cf);
101 double t_filter2 = ctkRealTimer();
102
103
104 double t_exec_pc2r1 = ctkRealTimer();
Detected potential issue
HMPP-ALT-FFT/VERSION1
/exec_D2Z_Z2D.c @line 95 - Advice54: A
call to the standard FFTW function
"fftw_execute" has been detected inside
a function.
Advice
Consider using an optimized library for
you application with the HMPP ALT
proxy.
CULA tools
Access to MyDevDeck web online resources
11 for (i = 1; i < M-1; ++i) // 2
12 {
13 for (j = 1; j < N -1; ++j) // 1
14 {
15 int a = rename(A, A);
16 A[i-1][j-1] = a ;
17 }
18 }
19 }
20
21 #pragma hmpp initLoop codelet, target=CUDA
22 void initLoop(int M, int N, real A[N][M])
23 {
24 int i, j;
25 for (i = 1; i < M-1; ++i) // 2
26 {
27 for (j = 1; j < N -1; ++j) // 1
28 {
29 A[i-1][j-1] = 3.14 ;
30 }
31 }
32 }
33
34 #pragma hmpp loopUnrolled codelet, target=CUDA
Detected potential issue
sample/data/src/mycode.c @line 25 -
Advice2: The computation density is low.
Loop Statistics
Number of array access: 1
Number of operations: 2 including 0 flops
Number of intrinsic operations: 0 including 0 flops
For more details, click here
Advice
The computation may fetch few

HMPP Performance Analyzer

- Dynamic analyses
- Synthetize metrics based on GPU execution profile



Tune your Codelet for Your Hardware

5 - Dealing with Libraries

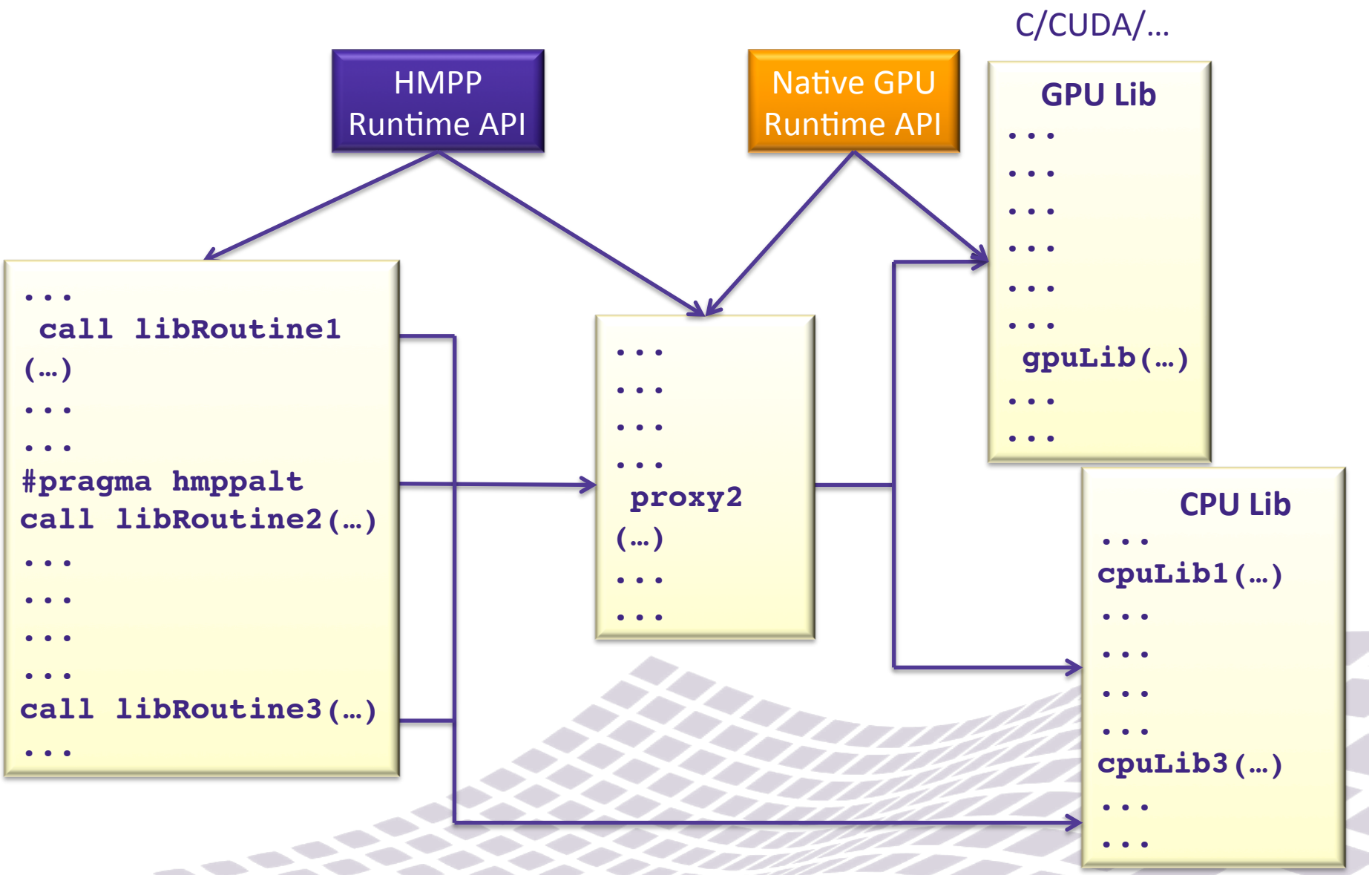
- Library calls can usually only be partially replaced
  - No one to one mapping between libraries (e.g.BLAS, FFTW, CuFFT, CULA,LibJacket)
  - No access to all code (i.e. avoid side effects)
  - Don't create dependencies on a specific target library as much as possible
  - **Still want a unique source code**

- Deal with multiple address spaces / multi-GPUs
  - Data location may not be unique (copies)
  - Usual library calls assume shared memory
  - Library efficiency depends on updated data location (long term effect)

- Libraries can be written in many different languages
  - CUDA, OpenCL, HMPP, etc.

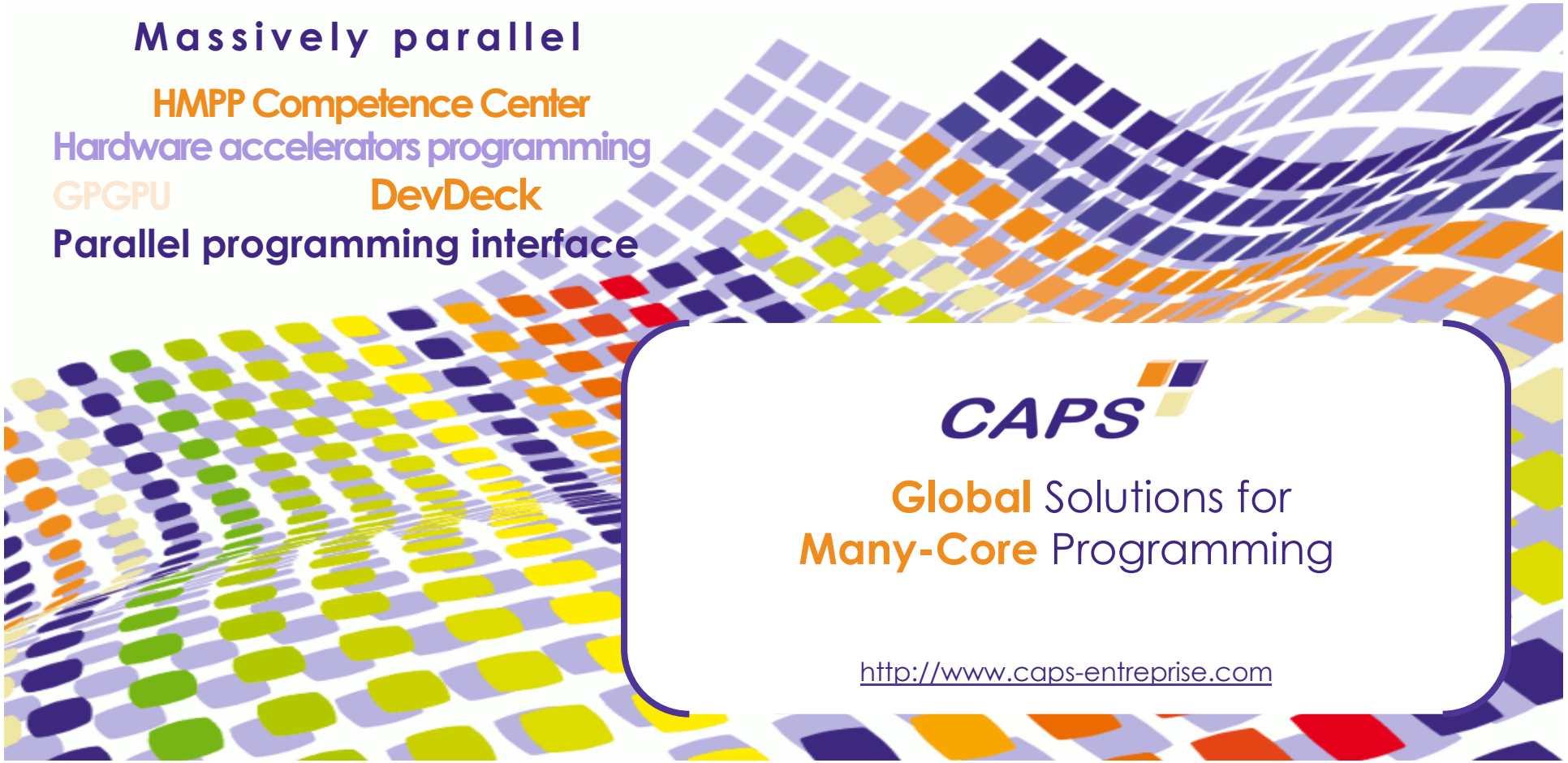
- There is not one binding choice depending on applications/users
  - Binding needs to adapt to uses depending on how it interacts with the remainder of the code
  - Choices depend on development methodology

Library Interoperability in HMPP 3.0



Conclusion

- Software has to expose massive parallelism
  - **The key to success is the algorithm!**
  - The implementation has “just” to keep parallelism flexible and easy to exploit
- Directive-based approaches are currently one of the most promising track
  - Preserve code assets
  - May separate parallelism exposure from the implementation (at node level)
- Remember that even if you are very careful in your choices you may have to rewrite parts of the code
  - Code architecture is the main asset here


Many-core programming Parallelization GPGPU NVIDIA Cuda OpenHMPP
CAPS
Directive-based programming Code Porting Methodology
OpenACC Hybrid Many-core Programmin HPC community Petaflops
Parallel computing HPC open standard Exaflops Open CL
High Performance Computing Code speedup Multi-core programming
Massively parallel
HMPP Competence Center
Hardware accelerators programming
GPGPU DevDeck
Parallel programming interface
CAPS
Global Solutions for
Many-Core Programming
http://www.caps-entreprise.com