Multi-core and Many-core for Preliminary Aircraft Design with Scilab

Q.V. Dinh
Dassault Aviation, R&D and Advanced Business

The H4H project

H4H = Hybrid for HPC

Optimise HPC applications on Heterogeneous Architectures

ITEA2 framework

- Start: October 1st, 2010. Duration: 3 years
- Partners from 4 countries:
  - France (8, Bull…+Scilab): Bull is the provider of HPC platform NOVA2
  - Germany (9, TUD)
  - Spain (4, Repsol)
  - Sweden (2, Efield/ESI)
- Since October 2012, inclusion of Perfcloud: + 4 French partners (CEA-DAM…)

Dassault Aviation participation

- Exploration of GPU technology (=/= code porting…)
- 5 use-cases:
  - Solution of sparse linear systems (2)
  - FEM Matrix and Residual assemblings on Unstructured Meshes (2)
  - **Scilab usage for “real time” treatment of numerical data (1)**

Motivation and objectives

In HPC, usually

- Seasoned HPC end-users and developers, responsible for the maintenance of our large HPC software (e.g. CFD)
- Performance is very important: aware of implementation issues
- Specialization: CFD, structures, electromagnetic…

With SciLAB, who are we targeting here ?

- Design engineers who, for their preliminary AC design work, use some of our "approximate" HPC software
- without much knowledge about the underlying complexity.
- "Real-time" feel: by combining approximation and performance
- Multidisciplinary

In short, what we have in mind is "HPC for the dummies", to help the AC design engineers boost their software performance, without asking them to spend man-months (years ?) to learn "HPC language"

SciGPGPU for EMM (1/5)

What is EMM ?

- Post-processing of Electro-Magnetic Measurements from anechoic chambers, to evaluate the far-field electro-magnetic properties of the AC
- Complex mathematical transformation involving 2d spline interpolations and FFTs.
- Coded into a Scilab module
- Ported on GPUs with the help of SciGPGPU.

SciGPGPU for EMM (2/5)

Basic “modus operandi” in SciGPGPU

- 2 types of variables are exposed to the user:
  - GPU variables : + functions to identify them
  - CPU variables : idem
- Simplified functions to go from one world to the other:
  - put(CPU->GPU), get(GPU->CPU), free(GPU memory) etc…
  - …without having to worry about addresses and types !
- Special GPU-adaptation for “hot” numerical kernels
  - FFTs, some BLAS functions, (spline-) interpolations etc…
  - …done by Scilab Enterprises, usually, on-demand services.
- GPU C-compiler for more interested GPU users
  - i.e. do-it-yourself GPU-adaptation.

SciGPGPU for EMM (3/5)

```
y=S';
x=A';

yyI=SInt;
xxI=AInt;

rH=real(H);
C = splin2d(y,x,rH,"natural");
rHI=interp2d(yyI,xxI,y,x,C);
iH=imag(H);
C = splin2d(y,x,iH,"natural");
iHI=interp2d(yyI,xxI,y,x,C);


timing(2)=timing(2)+toc(); // Interpolations...
```

```
gpuy=gpuTranspose(S);
gpux=gpuTranspose(A);
y=gpuGetData(gpuy); x=gpuGetData(gpux);
yyI=SInt;
xxI=AInt;
gpuyyI=gpuSetData(yyI); gpuxxI=gpuSetData(xxI);
rH=real(H);
C = splin2d(y,x,rH,"natural");
gpurHI=gpuInterp2d(gpuyyI,gpuxxI,gpuy,gpux,C,"C0");
iH=imag(H);
C = splin2d(y,x,iH, "natural");
gpuiHI=gpuInterp2d(gpuyyI,gpuxxI,gpuy,gpux,C,"C0");
gpuFree(gpuy); gpuFree(gpux);
gpuFree(gpuyyI); gpuFree(gpuxxI);
timing(2)=timing(2)+toc(); // Interpolations...
```

What are matrices S,A ?

- one 2d representation of - or "image of" - the complex electromagnetic field around the AC
- A measurement campaign has a large number of these images…

DASSAULT AVIATION

SciGPGPU for EMM (4/5)

Benchmark results

- For a 1000 x 1251 image
- GPU-equipped nodes of NOVA2 (*gpus* queue)
- SciGPGPU 1.3

	VAR3		VAR4		VAR5	VAR6
	cpu (s)	cpu+gpu (s)	cpu (s)	cpu+gpu (s)	cpu+gpu (s)	cpu+gpu (s)
Grid set-ups (ndgrid…)	3.03 **(1.0)**	3.02 **(1.0)**	3.03 **(1.0)**	3.02 **(1.0)**	3.30 **(0.9)**	2.95 **(1.02)**
Interpolation (splin2d, interp2d…)	2.48 **(1.0)**	1.37 **(1.8)**	2.49 **(1.0)**	1.37 **(1.8)**	0.58 **(4.3)**	0.89 **(2.8)**
Double => Complex	0.87 **(1.0)**	0.03 **(>10)**	0.88 **(1.0)**	0.01 **(>10)**	0.01 **(>10)**	0.01 **(>10)**
FFTs & shifts	5.83 **(1.0)**	1.49 **(3.9)**	4.72 **(1.2)**	0.56 **(10)**	0.69 **(8.4)**	0.56 **(10)**
Complex => Double & floor	2.59 **(1.0)**	2.22 **(1.2)**	0.00 **(>10)**	0.00 **(>10)**	0.00 **(>10)**	0.00 **(>10)**

SciGPGPU for EMM (5/5)

```
AInt = (180/%pi)*atan(real(-Ky),real(-Kx));
```

```
__global__ void KernelAtan(double* kx, double* ky, double Pi, double* out, int numberOfElement)
{
int iPos = threadIdx.x + blockIdx.x * blockDim.x;
if(iPos < numberOfElement)
{
out[iPos] = (180 / Pi) * (atan2(-ky[iPos], -kx[iPos]));
}
}
```

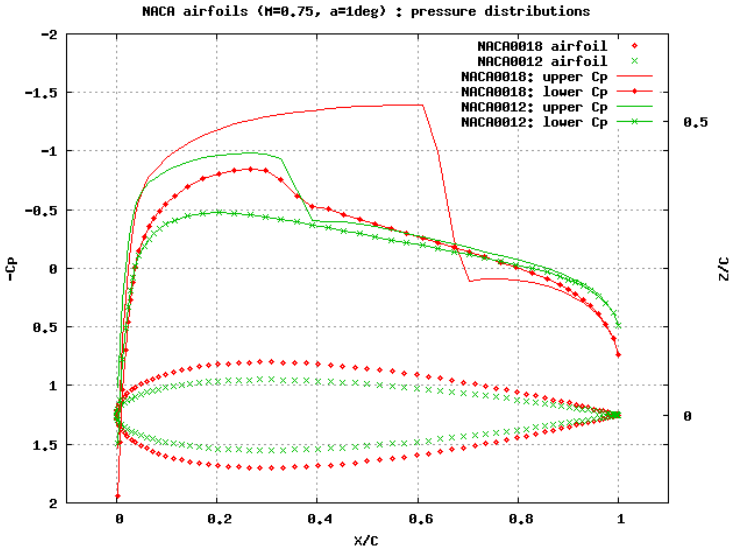
Final results for a 1000 x 1251 image

- A overall speed-up of more than 6
- Total elapsed time: 13 s -> less than 2 s

Scilab/MPI for Inverse design optimization in CFD (1/5)

What is Inverse design optimization ?

- Try to look for a shape with some desired aerodynamic properties…
- With the help of TSFOIL, an "approximate" CFD code




```
printf('Angle of attack       : alpha   = %9.6e\n',alpha);
printf('Free-stream Mach number: mach    = %9.6e\n',mach);
printf('Max. # of iterations   : maxit   = %9d\n',maxit);
tic();
[lift,drag,cm,xp,cpu,cpl,emu,eml,residual]=TSFOIL(alpha,mach,prof,cli,maxit);
tsf(1)=toc();
printf('Level of convergence   : residual= %5.3e\n',residual);
```

Scilab/MPI for Inverse design optimization in CFD (2/5)

To carry out the optimization,

- Usage of gradient-based optimizers
- Evaluation of gradients via finite-difference formulas

```
for optvar=1:optnvar
 prof(optvar,2) = prof(optvar,2) + dy
[liftp,dragp,cmp,xp,cpu,cpl,emu,eml,residual]=TSFOIL(alpha,mach,prof,cli,maxit);
 prof(optvar,2) = prof(optvar,2) - 2.*dy
[liftm,dragm,cmm,xp,cpu,cpl,emu,eml,residual]=TSFOIL(alpha,mach,prof,cli,maxit);
 prof(optvar,2) = prof(optvar,2) + dy
//
 grad(optvar,1) = (liftp - liftm)/(2.*dy)
 grad(optvar,2) = (dragp - dragm)/(2.*dy)
 grad(optvar,3) = (cmp   - cmm  )/(2.*dy)
end
```

Completely independent calls to TSFOIL

- Typical case of workflow distribution
- Easy to implement in the MPI paradigm

Scilab/MPI for Inverse design optimization in CFD (3/5)

Basic "modus operandi" with Scilab/MPI

- Workflow parallelism (and not data parallelism)
  - Assumption 1: good knowledge of (CFD) function inputs and outputs
  - Assumption 2: no knowledge of how inputs => inputs in function
    - e.g. no partitioning of complex data structures.
- As for all MPI models:
  - 1 Scilab process (= 1 MPI process) = 1 process id
- Simplified MPI primitives
  - Initialize, finalize, send, recv, broadcast, barrier etc…
  - …without having to worry about addresses, types and message ids !

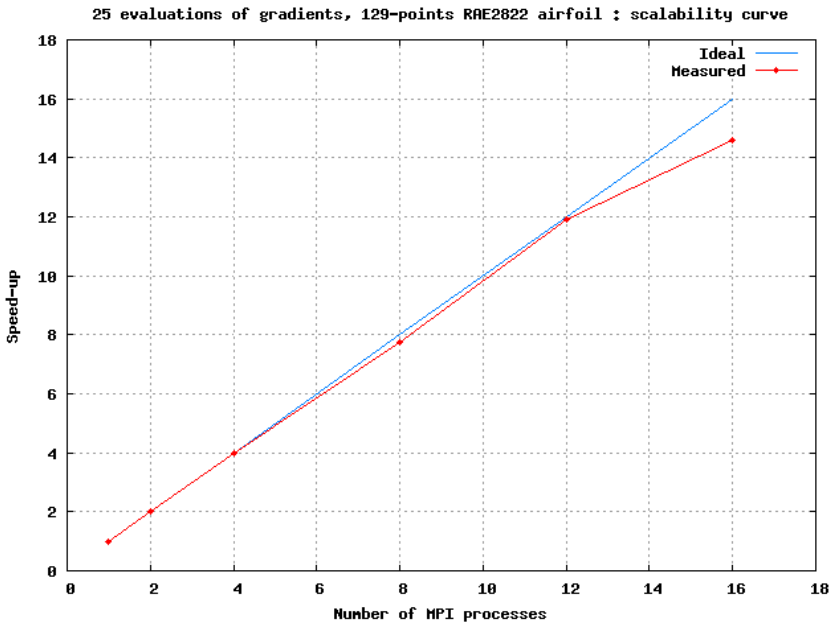
Scilab/MPI for Inverse design optimization in CFD (4/5)

```
for optvar=optvbnd(1):optvbnd(2)
 prof(optvar,2) = prof(optvar,2) + dy
[liftp,dragp,cmp,xp,cpu,cpl,emu,eml,residual]=TSFOIL(alpha,mach,prof,cli,maxit);
 prof(optvar,2) = prof(optvar,2) - 2.*dy
[liftm,dragm,cmm,xp,cpu,cpl,emu,eml,residual]=TSFOIL(alpha,mach,prof,cli,maxit);
 prof(optvar,2) = prof(optvar,2) + dy
//
 grad(optvar,1) = (liftp - liftm)/(2.*dy)
 grad(optvar,2) = (dragp - dragm)/(2.*dy)
 grad(optvar,3) = (cmp   - cmm  )/(2.*dy)
end
 // MPI communications
 if Master then
  for rnk=0:mpsize-1
  if (rnk~=mpmast) then
   gradv=MPI_Recv(rnk,0);
   optv1=varload(1,rnk+1);
   optv2=varload(2,rnk+1);
   grad(optv1:optv2,1:3) = gradv(optv1:optv2,1:3)
  end
  end
 else
  MPI_Send(grad,mpmast);
 end
```

Scilab/MPI for Inverse design optimization in CFD (5/5)

Benchmark results

- NOVA2, 2 x 8-cores nodes
- 25 evaluations of gradients
- Number of MPI blocks=2 to 16



Final results for 16-MPI run

- Total elapsed time: 6 minutes -> 24 seconds

Conclusions

There are maybe better (i.e. more performant !)

- GPU implementations of high-level codes
- Multi-core implementation of MPI
- …

But, in SciGPGPU and Scilab/MPI, what we were looking (and got) are:

- Easiness of use
  - A quick learning curve for non-seasoned HPC users
- A shield from:
  - The complexity of the underlying hardware
  - The pace of hardware evolutions
- Of course, the fact that it's open-source software…
  - Scilab/MPI is distributed in the latest Scilab release 5.5.0
  - SciGPGPU: still a research-level software…
    - Since November 2013, added features in CUDA and OpenCL
- …provided by a well-established ISV !

Thank you for your attention !

Any question ?