Programming Support for Cell/BE Multiprocessor

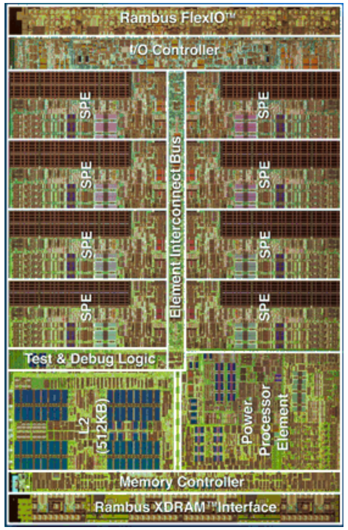
Enes Bajrovic and Eduard Mehofer

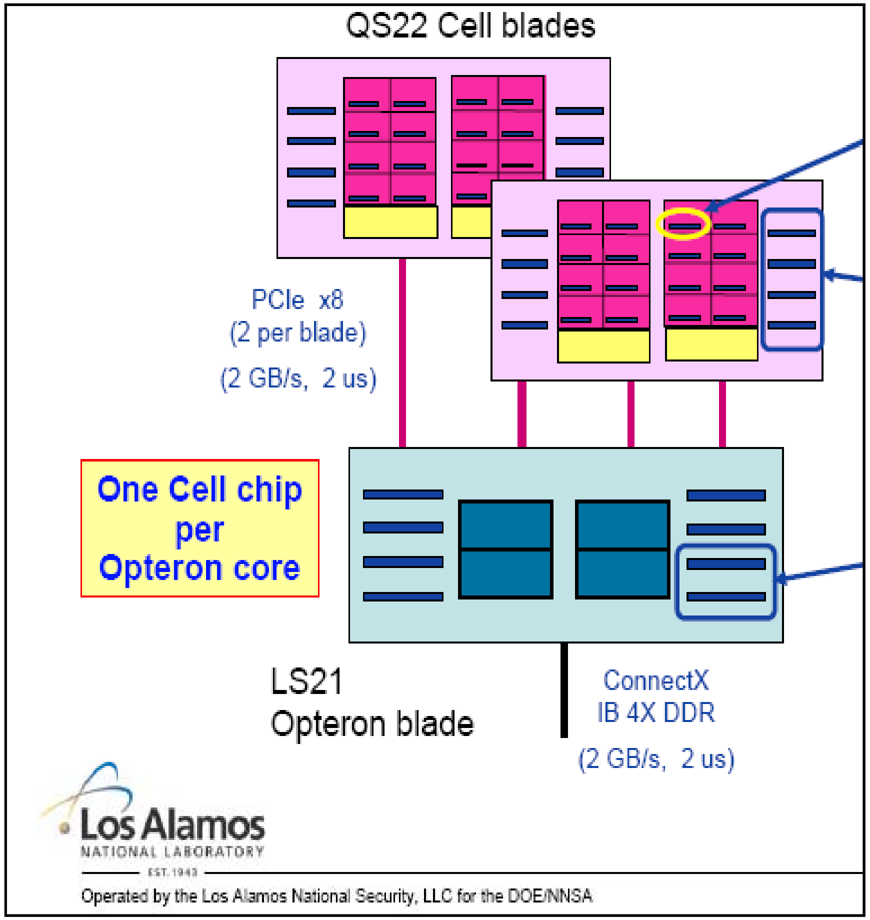
Department of Scientific Computing
University of Vienna

Motivation of Work

Architecture of current/future high-end computing systems

Application domain:
Scientific applications





New challenge: support for explicitly managed memory hierarchies.

History

Gelernter, Carriero. 1992

- Background: HPC dominated by message-passing and need for better programming support.

➢ **Coordination languages and their significance. Comm. ACM.**

- “diversity w.r.t. language, hardware platform, physical location … will be normal in the new era“.
- Coordination model and computation model separated.

Kennedy, Koelbel, Zima. 2007

➢ The rise and fall of HPF: an historical object lesson. ACM SIGPLAN on History of programming languages.

HPC still dominated by message-passing.

Programming approach: no consensus within community.

Efficiency: Interplay Between

1. Programmer

- controls parallelization explicitly (including data movements).

2. Parallelization framework VIECELL

- supports coordination model:
  - thread creation
  - work distribution
  - data movements

3. Native compiler

- compiles computation model for computing device and performs optimizations:
  vectorization, loop unrolling, software pipeling, etc.

Programming Framework ViECell

Design Principles

1. Processors.

- Stream architectures with accelerators and explicitly managed memory like Cell/BE.

2. Applications.

- Stream-like applications in computational science.

3. Program development

- Semantically equivalent sequential version of program available (all tools usable).

Example: Matrix-Vector Multiplication

PPU user code:

```
float A[M][N],X[N],Y[M];
… sequential execution
#pragma vie parallel
for (int i=0;i<M;i++)
   SPU_dot_pr(&A[i][0],X,&Y[i]);
… sequential executiuon
```

SPU user code:

```
#pragma vie public vec1(in,N),vec2(in,N),vec3(out,1)
void SPU_dot_pr(float vec1[],float vec2[],
                float vec3[])
{ float sum=0;
  for (int j=0;j<N;j++) {
     sum+=vec1[j]*vec2[j]; }
  vec3[0]=sum;
}
```

Note: parallelism hardware independent!

Problems Handled by Parallelization Framework

Matrix-vector multiplication:

- SPU function loaded / called only once (row-by-row streaming)
- double buffering optimization
- split blocks to fit in small memory (stream in / stream out)
- aggregation of small transfers
- second parameter only once to SPU

Native Compiler and Opt: Ex. Vector Add (1)

Scalar code:

(Single SPU)

	Gflops
GCC	~0.13
XLC	~3.72

```
for (i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
}
```

Native Compiler and Opt: Ex. Vector Add (2)

Vector code:

(Single SPU)

	Gflops	Speedup
GCC	~0.78	**6.00**
XLC	~3.72	1.00

```
vector float a[n], b[n], c[n];

for (i = 0; i < n/4; i++) {
     c[i] = spu_add(a[i], b[i]);
}
```

Native Compiler and Opt: Ex. Vector Add (3)

Several optimizations:

(Single SPU)

	Gflops (unroll 2)	Gflops (unroll 6)	Speedup
GCC	~1.71	~3.81	**12.15** / **30**
XLC	~3.72	~3.84	1.00 / 1.03

```
vector float x0,x1,x2,x3,x4,x5;
vector float y0,y1,y2,y3,y4,y5;
vector float z0,z1,z2,z3,z4,z5;
...
for (i = 0; i < n/4 - 2; i+=6) {
    // Store [i] - [i+5]
    c[i+0] = z0; c[i+1] = z1; c[i+2]=z2;
    c[i+3] = z3; c[i+4] = z4; c[i+5]=z5;
    // Compute [i+1] – [i+6]
    z0=spu_add(x0, y0); z1=spu_add(x1, y1); z2=spu_add(x2, y2);
    z3=spu_add(x3, y3); z2=spu_add(x2, y2); z3=spu_add(x3, y3);
    // Load next a: [i+12] - [i+17]
    x0 = a[i+2]; x1 = a[i+3]; x2 = a[i+4];
    x3 = a[i+5]; x4 = a[i+6]; x5 = a[i+7];
    // Load next b: [i+12] - [i+17]
    y0 = b[i+2]; y1 = b[i+3]; y2 = b[i+4];
    y3 = b[i+5]; y4 = b[i+6]; y5 = b[i+7];
}
```

Related Work

Graphics Community

- OpenCL
- Cuda (NVIDIA)
- Brook+ (AMD)

HPC Community

- OpenMP: with extensions
- PGAS: CAF, UPC, Titanium
- DARPA HPCS Program: High Productivity Computing Systems (High Performance Software Crisis – ends 2010).
  X10 (IBM), Chapel (Cray), Fortress (Sun)
- Sequoia

Conclusion / Future Work

- For efficiency:

  interplay

  programmer — parallelization framework — native compiler

  (assign mangageable tasks only)

- Programmer: explicit parallel programming
- Separation coordination model and computation model
- Future work:
  - Move on to GPUs.