



Enabling Grids for E-science

CPU Architectures & Compilers

Vladimir Slavnic
slavnic@scl.rs

Scientific Computing Laboratory
Institute of Physics Belgrade, Serbia



SEE-GRID-SCI
SEE-GRID infrastructure for regional eScience



Oct. 07, 2009

www.eu-egee.org



Information Society



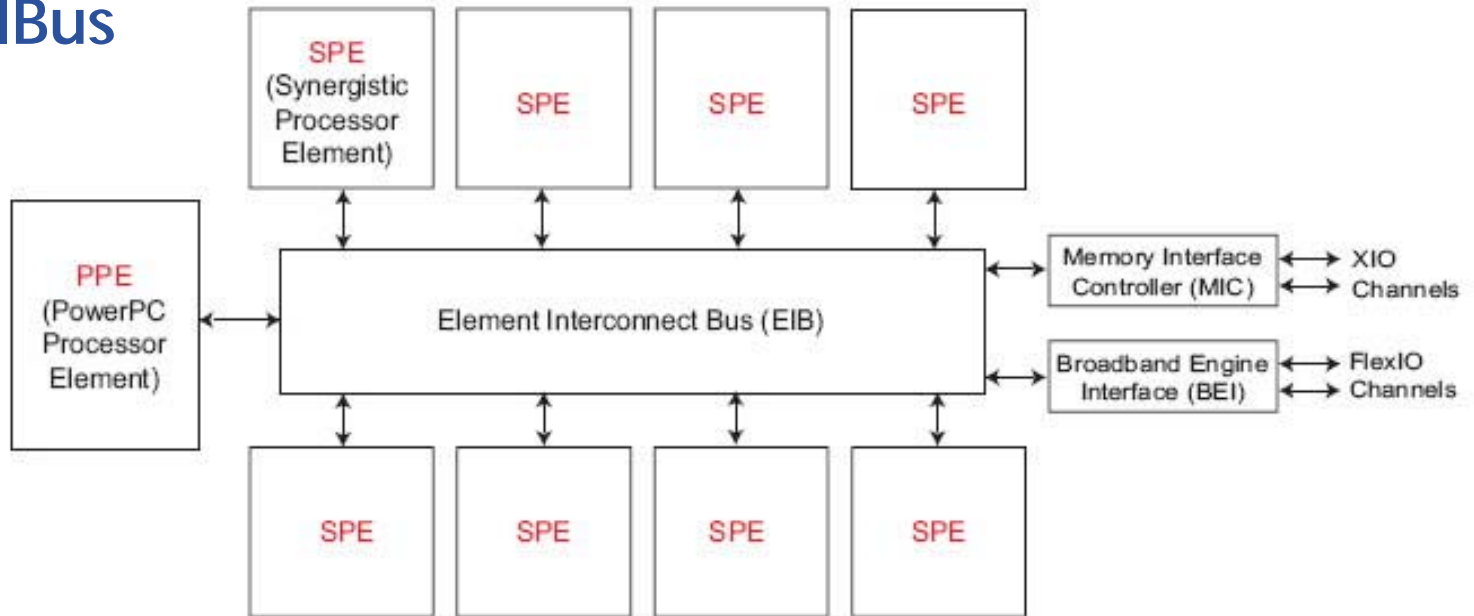
- CPU Architectures showcase
 - Intel Xeon (x86)
 - POWER6 (PPC)
 - Cell BE (PPC + SPU)
- Programming examples
- GNU, XLC and ICC compilers
- Results of practical application

- Developed by Intel
- Streaming Single Instruction Multiple Data (SIMD) Extensions SSE2, SSE3, SSE4.1
- Support Intel® 64 Architecture and Intel VT
- Up to four cores (8 cores soon)
- Multithreading announced for Nehalem
- SGI PLEIADES #4 TOP500.org
- Juropa #10 TOP500.org
- Various servers and desktops

- Developed by IBM
- AltiVec unit
- Dual-core processor
- Two-way simultaneous multithreading (SMT)
- Up to 4.7GHz
- Implements ViVA-2, Virtual Vector Architecture
- IBM JS12 (two 3.8 GHz POWER6 cores) and JS22 (four 4.0 GHz cores) blade servers
- POWER6 systems: 520, 550 , 560, 570, 595 and water-cooled Power 575 (2U nodes with 32 POWER6 cores at 4.7 GHz with up to 256 GB of RAM)

- Developed by Sony Computer Entertainment, Toshiba, and IBM Cell BE (Broadband Engine)
- Heterogeneous architecture
- Original and improved version PowerXCell 8i
- Sony's PlayStation 3 game console
- IBM Roadrunner first supercomputer to run at petaFLOPS (#1 at TOP500.org)
- IBM QS21 and QS22 blade servers
- Toshiba's HDTVs using Cell
- PCI Express Board
- ~100 GFlops

- Architectural overview
 - 1 PPU
 - 8 SPEs
 - EIBus



- PPU (Power Processor Element)
 - General-purpose, dual-threaded, 64-bit RISC processor
 - Fully compliant with the 64-bit PowerPC Architecture, with the Vector/SIMD Multimedia Extension
 - Intended primarily for
 - Control processing
 - Running operating systems
 - Managing system resources
 - Managing SPE threads
 - Operating at 3.2 GHz

- SPU (Synergetic Processor Element)
 - Single-instruction, multiple-data (SIMD) processor elements that are meant to be used for data-rich operations allocated to them by the PPE
 - Not optimized for running operating system
 - SPE contains a RISC core, 256 KB software-controlled locale storage (LS) for instructions and data, 128-bit, 128-entry unified register file, MFC
 - DMA exclusively memory transfers to main memory and the local storage of other SPE's
 - Synergistic Processor Unit Instruction Set Architecture

No cache!

- Separate code for PPU and for SPEs
- Separate compilers
- Simple example
- PPU code

```
#include<stdio.h>
#include<libspe2.h>
#include<pthread.h>
void *ppu_thread_function(void *arg) {
    spe_context_ptr_t context = *(spe_context_ptr_t *) arg;
    unsigned int entry = SPE_DEFAULT_ENTRY;

    spe_context_run(context,&entry,0,NULL,NULL,
    NULL);

    pthread_exit(NULL); }
```

```
extern spe_program_handle_t hello_spu;
int main(void) {
    spe_context_ptr_t context;
    pthread_t pthread;
    context = spe_context_create(0,NULL);
    spe_program_load(context,&hello_spu);

    pthread_create(&pthread,NULL,&ppu_thread_function,&context);
    pthread_join(pthread,NULL);
    spe_context_destroy(context);
    printf ("Hello world ! PPU\n");
    return 0;
}
```

Spu code

```
#include<stdio.h>
int main(unsigned long long speid) {
    printf ("Hello world ! SPU\n");
    return 0;
}
```

- Loop summing optimization example
 - Unroll loop
 - Vectorize loop

```
// 16 iterations of a loop
int rolled_sum(unsigned char bytes[16])
{
    int i;
    int sum = 0;
    for (i = 0; i < 16; ++i)
    { sum += bytes[i]; }
    return sum;
}
```

- Loop summing example

- Unroll loop

```
// 4 iterations of a loop, with 4 additions in each iteration int
unrolled_sum(unsigned char bytes[16])
```

```
{
int i;
int sum[4] = {0, 0, 0, 0};
for (i = 0; i < 16; i += 4) {
sum[0] += bytes[i + 0];
sum[1] += bytes[i + 1];
sum[2] += bytes[i + 2];
sum[3] += bytes[i + 3]; }
return sum[0] + sum[1] + sum[2] + sum[3];
}
```

- Loop summing example

- Vectorize loop

```
// Vectorized for Vector/SIMD Multimedia Extension
int vectorized_sum(unsigned char bytes[16])
{ vector unsigned char vbytes;
  union {
    int i[4];
    vector signed int v;
  } sum;
  vector unsigned int zero = (vector unsigned int){0};
  // Perform a misaligned vector load of the 16 bytes.
  vbytes = vec_perm(vec_ld(0, bytes), vec_ld(16, bytes), vec_lvsl(0, bytes));
  // Sum the 16 bytes of the vector
  sum.v = vec_sums((vector signed int)vec_sum4s(vbytes, zero)
    (vector signed int)zero);
  // Extract the sum and return the result.
  return (sum.i[3]); }
```

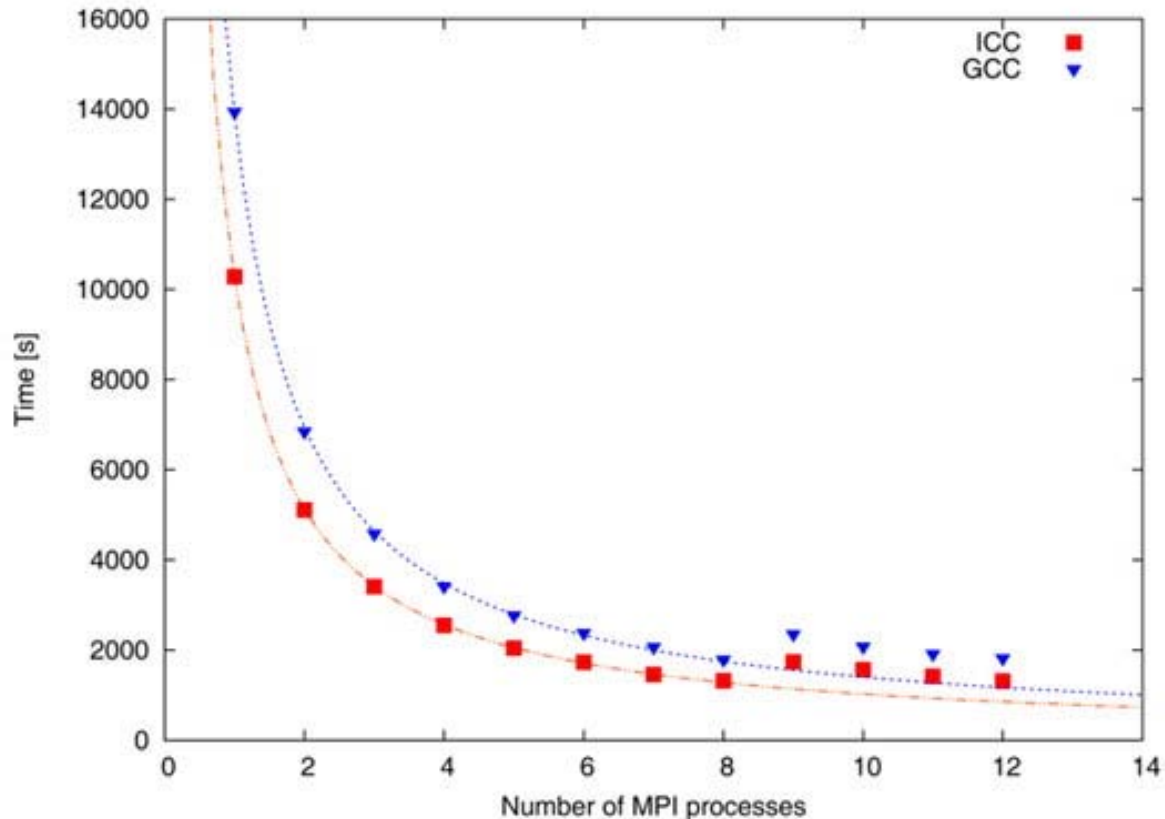
- **GCC (GNU Compiler Collection)**
 - Open source solution
 - Supports numerous architectures
 - Supports various operating systems
 - OpenMP support (since version 4.2)
 - Good user support through Community
- **ICC Intel® C++ Compiler Professional Edition**
 - Advanced optimization, multithreading, and processor support
 - Automatic processor dispatch, vectorization, and loop unrolling, OpenMP support
 - Commercial

- **IBM XL C/C++ Compiler (Standard and for Multicore Acceleration)**
 - Solution for POWER platform
 - Cross-compiler possibility
 - AltiVec API support
 - Provides automated SIMD capabilities
 - OpenMP support
 - ppuxlc or ppuxlc++ and SPU-specific commands spuxlc, spuxlC, spuxlc++ for Multicore version
 - Commercial

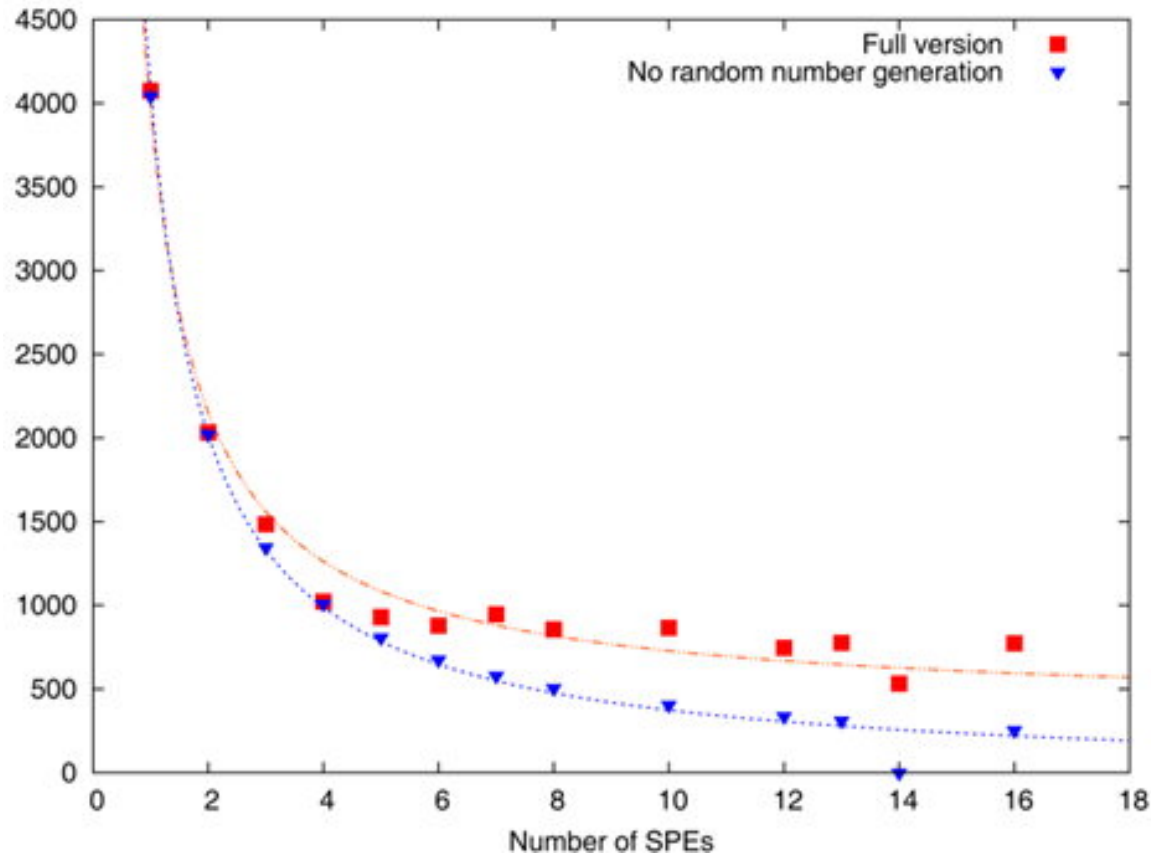
- Path integral Monte Carlo SPEEDUP code (SCL)
- 5120000 MC iterations
- Intel Xeon 5405 2.0 GHz, FSB of 1333MHZ , L2 cache of 12MB
- POWER6 4.0 GHz, 64 KB I-cache, 32 KB D-cache L1 per core, 4 MB L2 cache per core, L3 cache 32 MB
- Cell 3.2 GHz, 32/32 KB L1 (i/d) and 512 KB L2 cache

Compiler \ Platform	GCC	ICC	XLC
Intel	13760±50s	1630±30s	-
POWER6	17000±10s	-	1900±10s
Cell	49410±50s	-	14020±20s

- MPI version of SPEEDUP code
- ICC and GCC



- Cell version of SPEEDUP code
- XLC compiler used



- **GPGPU Programming**
 - Close to Metal, (Stream), AMD/ATI
 - CUDA (Compute Unified Device Architecture), Nvidia's GPGPU technology
 - DirectCompute Microsoft's GPU Computing API - Initially released with the DirectX 11 API
- Intel Larrabee
- OpenCL (Open Computing Language)
- ...?

- **POWER** <http://www-03.ibm.com/technology/power/>
- **XLC** <http://www-01.ibm.com/software/awdtools/xlcpp/>
- **GCC** <http://gcc.gnu.org/>
- **ICC** <http://software.intel.com/en-us/articles/non-commercial-software-development/>
- **Cell Sym** <http://www.alphaworks.ibm.com/tech/cellsystemsimg>