Tools for Multi-Cores and Multi-Targets

Sebastian Pop

Advanced Micro Devices, Austin, Texas

The Linux Foundation Collaboration Summit
April 7, 2011

Tools for multi-cores

Why the multi-cores trend?

- more transistors (less power) $\Rightarrow$ more cores $\Rightarrow$ more tasks in parallel

Tools:

- POSIX threads
- OpenMP: directives for C, C++, and Fortran
- automatic parallelization with Graphite

Tools for multi-targets

Why the multi-target trend?

- general purpose multi-cores are not as performant as heterogeneous accelerators tuned for a specific workload (graphics, network, DSP, co-processors, etc.)

Tools:

- compiling across different targets with GCC
- OpenCL
- OpenMP extensions for accelerators
- automatic parallelization with Graphite-OpenCL

Part 1: tools for multi-cores

- POSIX threads
- OpenMP: directives for C, C++, and Fortran
- automatic parallelization with Graphite

POSIX threads

- library to program shared memory multi processors
- portable: POSIX standard avoids OS specific APIs

Pthreads example

```c
#include <pthread.h>
#include <stdio.h>
#define NTHREADS 4
#define N 1000000
#define ITERATIONS N / NTHREADS
int sum=0, a[N];
pthread_mutex_t sum_mutex;

void *do_work(void *tid)
{
  int mysum=0;
  int *mytid = (int *) tid;
  int start = (*mytid * ITERATIONS);
  int end = start + ITERATIONS;
  int i;
  for (i=start; i < end ; i++) {
    a[i] = i;
    mysum += a[i];
  }
  pthread_mutex_lock (&sum_mutex);
  sum += mysum;
  pthread_mutex_unlock (&sum_mutex);
  pthread_exit(NULL);
}
```

```c
int main(int argc, char *argv[])
{
  int i, start, tids[NTHREADS];
  pthread_t threads[NTHREADS];
  pthread_attr_t attr;

  pthread_mutex_init(&sum_mutex, NULL);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr,
          PTHREAD_CREATE_JOINABLE);
  for (i=0; i<NTHREADS; i++) {
    tids[i] = i;
    pthread_create(&threads[i], &attr, do_work,
                   (void *) &tids[i]);
  }

  for (i=0; i<NTHREADS; i++)
    pthread_join(threads[i], NULL);
  printf ("sum=%d\n", sum);

  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&sum_mutex);
  pthread_exit (NULL);
}
```

```
# gcc -pthread sum.c -o sum
# ./sum
```

https://computing.llnl.gov/tutorials/pthreads/samples/arrayloops.c

OpenMP (Open Multi-Processing)

- portable (GCC 4.2 or later, XLC, SunCC, ICC, PGI, Fujitsu, Pathscale, HP, MS, Cray)
- explicit parallelization of shared memory multi-threads
  - compiler directives for C, C++, Fortran
  - runtime library routines
  - environment variables
- incrementally parallelize a serial program
- OpenMP3.0 introduced task parallelism (not for HPC)
- OpenMP3.1 extends atomic operations (not for HPC)
- OpenMP4.0 adds error and exceptions handling (not for HPC)

OpenMP example

```
#include <stdio.h>
#define NTHREADS 4
#define N 1000000
int sum=0, a[N];

int main(int argc, char *argv[])
{
  int i;
#pragma omp parallel for reduction(+:sum) num_threads(NTHREADS)
  for (i=0; i<N; i++){
    a[i] = i;
    sum = sum + a[i];
  }
  printf("sum=%d\n", sum);
  return 0;
}
```

```
# gcc -fopenmp sum.c -o sum
# ./sum
```

Automatic parallelization with Graphite

```c
#include <stdio.h>
#define N 1000000
int sum=0, a[N];

int main(int argc, char *argv[])
{
  int i;
  for (i=0; i<N; i++) {
    a[i] = i;
    sum += a[i];
  }
  printf("sum=%d\n", sum);
}
```

```
# gcc -Ofast -ftree-parallelize-loops=4 -floop-parallelize-all sum.c -o sum
# ./sum
```

- from sequential code, detect parallel loops
- generate OpenMP parallel loops
- in GCC 4.5, 4.6, ...

Part 2: tools for multi-targets

- compiling across different targets with a GCC toolchain
- OpenCL
- OpenMP extensions for accelerators
- Graphite-OpenCL

GCC Toolchain

- binutils: assembler, linker, debugger, etc
- glibc, uclibc: C standard library
- gcc: GNU Compiler Collection

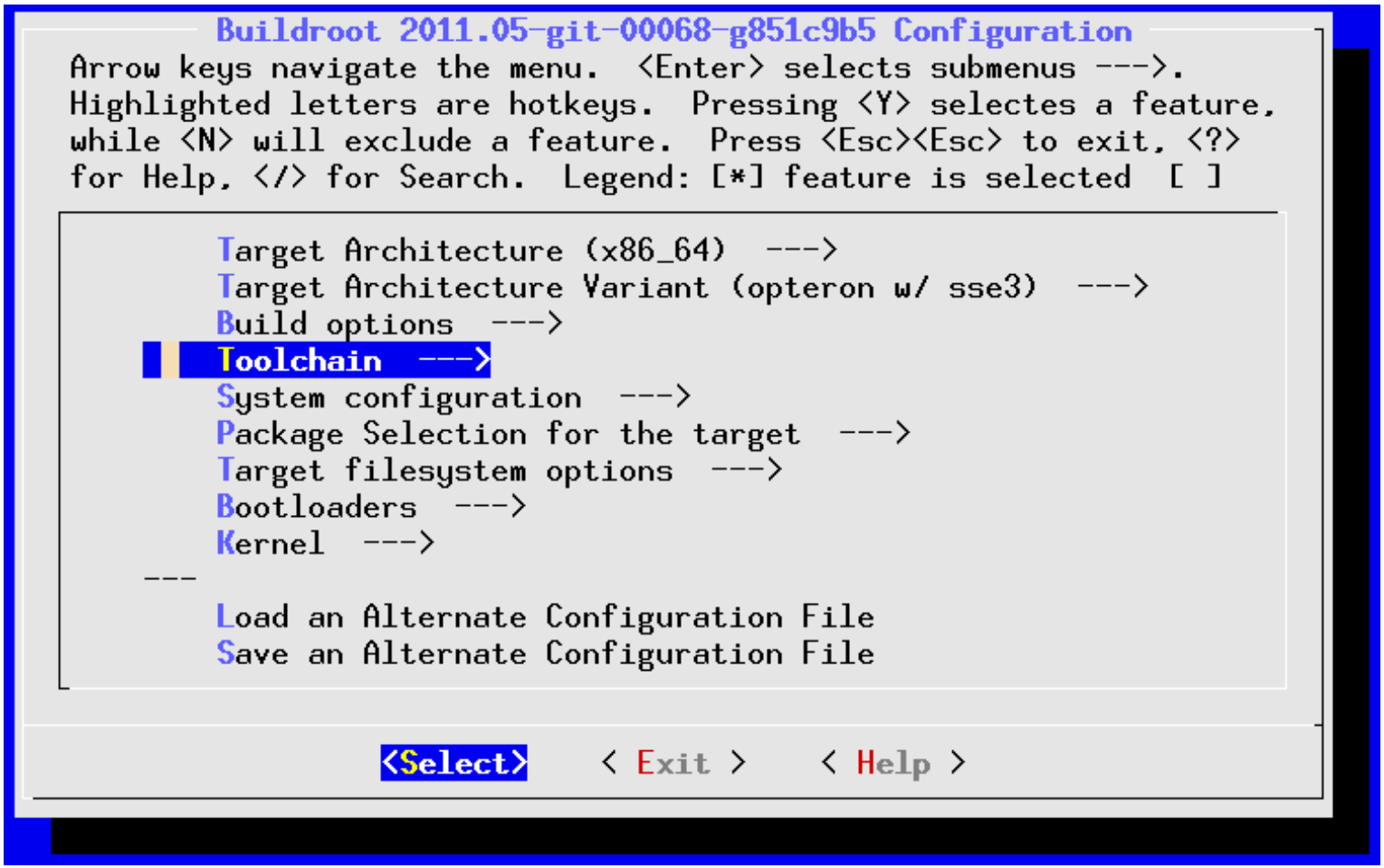
Building a toolchain

```
# git clone git://sourceware.org/git/binutils.git
# ../binutils/configure --build=b --host=h --target=t
# make && make install

# git clone git://gcc.gnu.org/git/gcc.git
# cd build-gcc
# ../gcc/configure --build=b --host=h --target=t
# make all-gcc && make install-gcc

# git clone git://sourceware.org/git/glibc.git
# ../glibc/configure --build=b --host=t
# make && make install

# cd build-gcc
# make && make install
```

- native: $b = h = t$
- cross: $b = h \neq t$

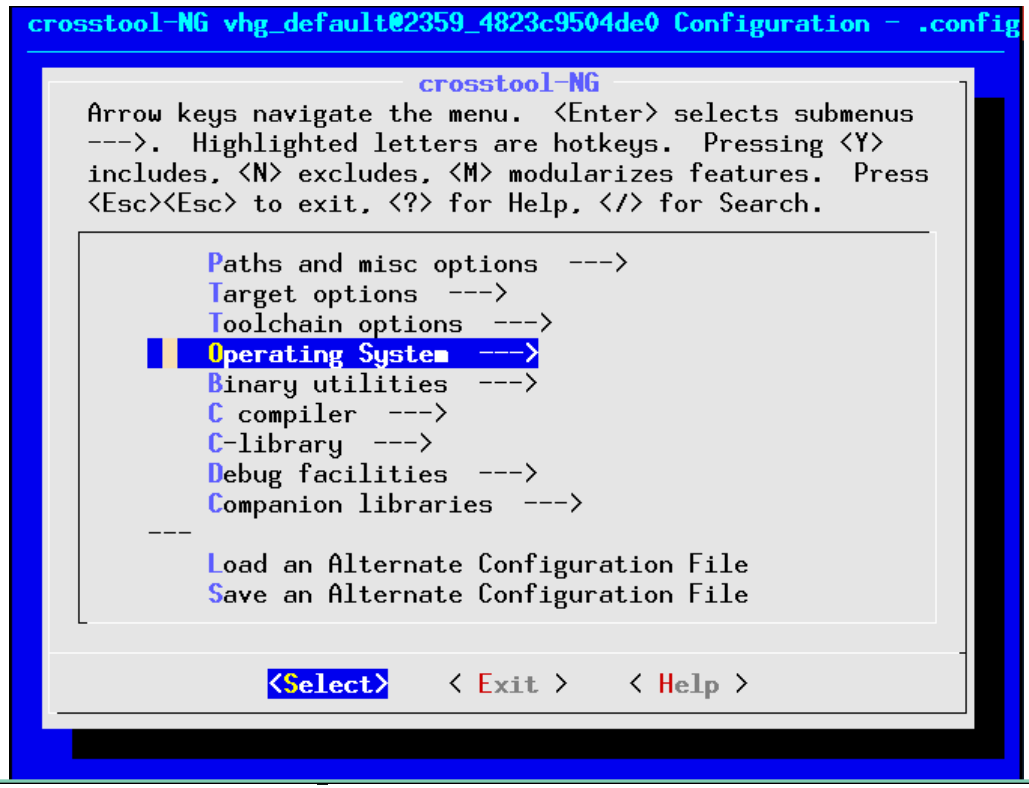
http://buildroot.org

```
# git clone git://git.buildroot.net/buildroot
# cd buildroot
# make menuconfig
# make
```



http://ymorin.is-a-geek.org/projects/crosstool

```
# hg clone http://ymorin.is-a-geek.org/hg/crosstool-ng
# ./configure --prefix=/some/place
# make
# make install
# export PATH=/some/place/bin
# cd /your/development/directory
# ct-ng menuconfig
# ct-ng build
```



Split compilation: two targets PowerPC and Cell SPE

```c
#include <stdio.h>
#include <libspe2.h>
#define N 1000000
int sum=0, a[N];
struct data {unsigned long a, sum;} d;
int main()
{
  spe_stop_info_t x;
  unsigned int entry = SPE_DEFAULT_ENTRY;
  spe_program_handle_t *p=spe_image_open("spe");
  spe_context_ptr_t c=spe_context_create(0, 0);
  spe_program_load(c, p);
  d.n = N;
  d.a = (unsigned long) a;
  d.sum = (unsigned long) sum;
  spe_context_run(c, &entry, 0, &d, NULL, &x);
  spe_context_destroy(c);
  spe_image_close(p);
  printf ("sum=%d\n", d.sum);
}
```

```
# gcc -lspe2 sum.c -o sum
```

```c
#define N 1000000
int a_spe[N];
struct data {unsigned long a,sum;} d;
int main(unsigned long long spe,
         unsigned long long argp,
         unsigned long long envp)
{
  int i, tag = 1;
  spu_mfcdma64(&d,mfc_ea2h(argp),mfc_ea2l(argp),
    sizeof(struct data),tag,MFC_GET_CMD);
  spu_writech(MFC_WrTagMask, 1 << tag);
  spu_mfcstat(MFC_TAG_UPDATE_ALL);
  spu_mfcdma64(a_spe,mfc_ea2h(d.a),mfc_ea2l(d.a),
        N*sizeof(int),tag,MFC_GET_CMD);
  spu_writech(MFC_WrTagMask, 1 << tag);
  spu_mfcstat(MFC_TAG_UPDATE_ALL);
  for (i=0; i<N; i++) {
    a_spe[i] = i*1.0;
    d.sum += a_spe[i];
  }
  spu_mfcdma64(a_spe,mfc_ea2h(d.a),mfc_ea2l(d.a),
        N*sizeof(int),tag,MFC_PUT_CMD);
  spu_writech(MFC_WrTagMask, 1 << tag);
  spu_mfcstat(MFC_TAG_UPDATE_ALL);
}
```

```
# spu-gcc spe.c -o spe
```

```
# ./sum
```

http://www.kernel.org/pub/linux/kernel/people/geoff/cell/ps3-linux-docs/ps3-linux-docs-latest/

CellProgrammingPrimer.html

OpenCL (Open Computing Language)

- portable: OpenCL is an open standard
- library and compilers targeting heterogeneous architectures
- data parallel: for each data point, execute kernel
- task parallel: work queues
- as easy (difficult) to program as POSIX threads

OpenCL example

```
#include <CL/cl.h>
#include <stdio.h>
#define N 1000000
const char *src = "__kernel void "
  "reduce(__global uint4*i, __global uint4*o,"
  "       __local uint4 *sdata) {"
  "    unsigned int tid = get_local_id(0);"
  "    unsigned int bid = get_group_id(0);"
  "    unsigned int gid = get_global_id(0);"
  "    unsigned int l = get_local_size(0);"
  "    sdata[tid] = i[gid];"
  "    barrier(CLK_LOCAL_MEM_FENCE);"
  "    for(int s=l/2; s>0; s>> =1) {"
  "      if(tid < s) "
  "         sdata[tid] += sdata[tid + s];"
  "      barrier(CLK_LOCAL_MEM_FENCE);"
  "    }"
  "    if(tid == 0) o[bid] = sdata[0];}";
int main () {
  cl_int x; cl_platform_id p; cl_device_id d;
  int groupSz = 256, vectorSize = 4;
  int mulFactor = groupSz * vectorSize;
  int i, len = (N / mulFactor) * mulFactor;
  int nb = len / groupSz;
  int *input=(int*)malloc(N*sizeof(cl_uint4));
  for(i = 0; i < N; ++i) input[i] = i;
  clGetPlatformIDs(1,&p,0);
  clGetDeviceIDs(p,CL_DEVICE_TYPE_CPU,1,&d,0);
  cl_context c=clCreateContext(0,1,&d,0,0,&x);
  cl_command_queue q=clCreateCommandQueue
   (c,d,0,&x);
  cl_mem b=clCreateBuffer(c,CL_MEM_READ_ONLY
   |CL_MEM_COPY_HOST_PTR,len*sizeof(cl_uint4),
   input,&x);
  cl_program prog = clCreateProgramWithSource
   (c,1,&src,0,&x);
  clBuildProgram(prog,1,&d,0,0,0);
  cl_kernel k = clCreateKernel(prog,"reduce",&x);
  cl_uint*t=(cl_uint*)malloc(nb*sizeof(cl_uint4));
  cl_mem out = clCreateBuffer(c,CL_MEM_WRITE_ONLY
   |CL_MEM_USE_HOST_PTR,nb*sizeof(cl_uint4),t,&x);
  clSetKernelArg(k,0,sizeof(cl_mem),(void*)&b);
  clSetKernelArg(k,1,sizeof(cl_mem),(void*)&out);
  clSetKernelArg(k,2,groupSz*sizeof(cl_uint4),0);
  size_t globalThreads[] = {len};
  size_t localThreads[] = {groupSz};
  clGetKernelWorkGroupInfo(k,d,
   CL_KERNEL_LOCAL_MEM_SIZE,sizeof(cl_ulong),0,0);
  clEnqueueNDRangeKernel(q,k,1,0,globalThreads,
      localThreads,0,0,0);
  clFinish(q);
  clEnqueueReadBuffer(q,out,CL_TRUE,0,
   nb*sizeof(cl_uint4),t,0,0,0);
  cl_uint sum = 0;
  for(i = 0; i < nb * vectorSize; ++i)
    sum += t[i];
  printf("sum=%d\n", sum);
  clReleaseMemObject(out);clReleaseKernel(k);
  clReleaseProgram(prog);clReleaseMemObject(b);
  clReleaseCommandQueue(q);clReleaseContext(c);
  return 0;}
```

```
# gcc -lOpenCL sum.c -o sum
# ./sum
```

OpenMP extensions for accelerators

- new directives:
  - acc_region, acc_loop, acc_region_loop, acc_barrier
  - acc_call, acc_call_declaration, acc_call_definition
  - acc_data, acc_update, acc_res, acc_mirror
- extensions not yet reviewed by OpenMP language committee
- will be submitted for inclusion in OpenMP 4.0

Example of OpenMP accelerated loop

```
#include <stdio.h>
#define N 1000000
int sum=0.0, a[N];

int main()
{
  int i;
#pragma omp acc_loop reduction(+:sum)
  for (i=0; i<N; i++) {
    a[i] = i;
    sum += a[i];
  }
  printf("sum=%d\n", sum);
}
```

```
# gcc -fopenmp sum.c -o sum
# ./sum
```

Automatic parallelization with Graphite-OpenCL

```
#include <stdio.h>
#define N 1000000
int sum=0, a[N];

int main(int argc, char *argv[])
{
  int i;
  for (i=0; i<N; i++) {
    a[i] = i;
    sum += a[i];
  }
  printf("sum=%d\n", sum);
}
```

```
# gcc -Ofast -fgraphite-opencl sum.c -o sum
# ./sum
```

- from sequential code, detect parallel loops
- generate OpenCL code
- will probably be in GCC 4.7 (sometime in 2012)

Call for collaboration

The GCC toolchain **should not depend** on proprietary libs.
We will need a free open source implementation of OpenCL.

Conclusion

- for multi-cores
  - write OpenMP code instead of POSIX threads
  - use GCC: it supports OpenMP 3.0
- for multi-targets
  - use GCC: it targets most of the processors
  - we will need a free open source implementation of OpenCL
  - use OpenCL (for now): it is an open standard
  - programming with OpenMP 4.0 will be easier than OpenCL