A PORTABLE OPENMP RUNTIME LIBRARY BASED ON MCA APIS FOR MULTICORE EMBEDDED SYSTEMS

Sunita Chandrasekaran (sunita@cs.uh.edu)
Cheng Wang
Barbara Chapman
HPCTools Group, University of Houston, USA

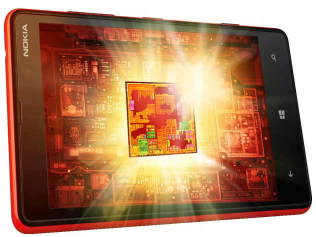
IN COLLABORATION WITH FREESCALE SEMICONDUCTOR (FSL) AND SEMICONDUCTOR RESEARCH CORPORATION (SRC)

OpenMP Booth @ SC, November 2013

Automation
Copier, Fax machines,
printers, scanners,
multi-function peripherals,
point of sale terminals,
storage devices, smartcards

Consumer Electronics
Music players, digital cameras,
DVD players, set-top boxes,
PDAs, videogames,
GPS receivers, home appliances

Medical Electronics
Patient monitoring,
surgical systems,
diagnostic equipment,
imaging, electronic stethoscopes



Telecom / Datacom
Routers, switches, bridges,
cellular phones, smart devices,
networking gateways

Embedded Systems

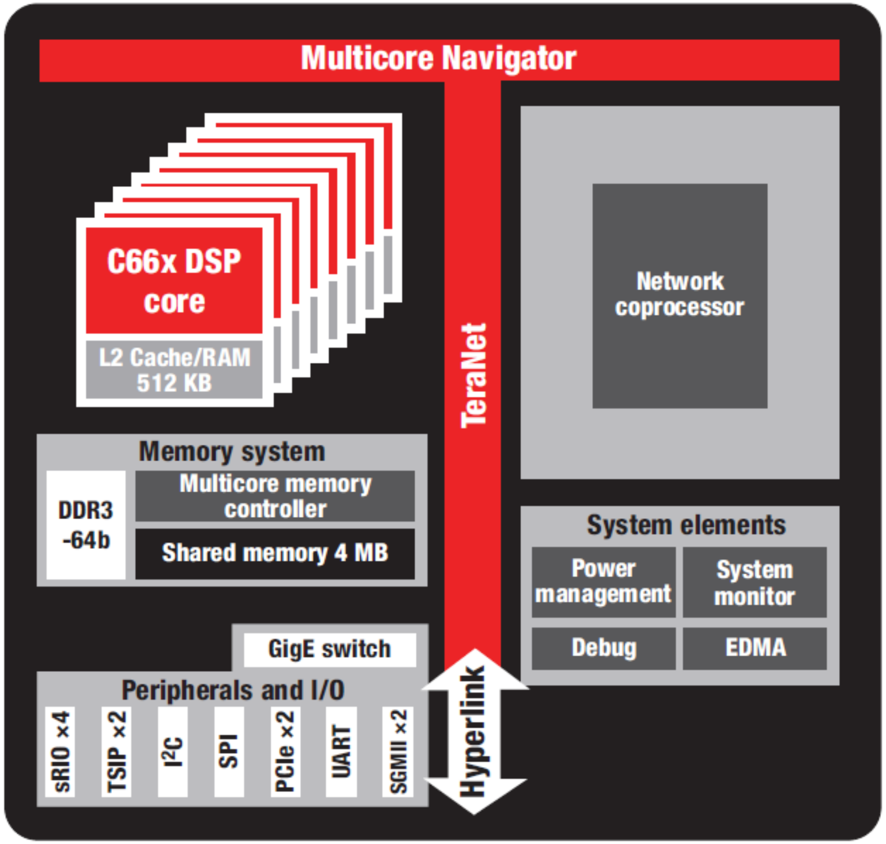
Remote Automation
Building automation
e.g. heating, ventilation,
air-conditioning (HVAC),
home automation, utility meters

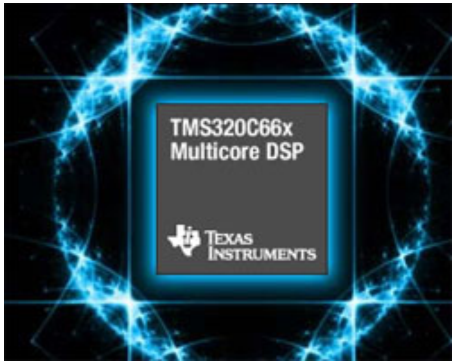
Military / Aerospace
Satellite systems, radar,
sonar, navigation,
weather systems,
flight control systems,
aircraft management systems

Industrial Controls
Smart sensors,
special purpose controllers,
networking, process controls

Automotive Electronics
Electronic control units
used in chassis, body electronics,
security, power train,
in-vehicle entertainment,
and infotainment systems

Multicore in Embedded Systems

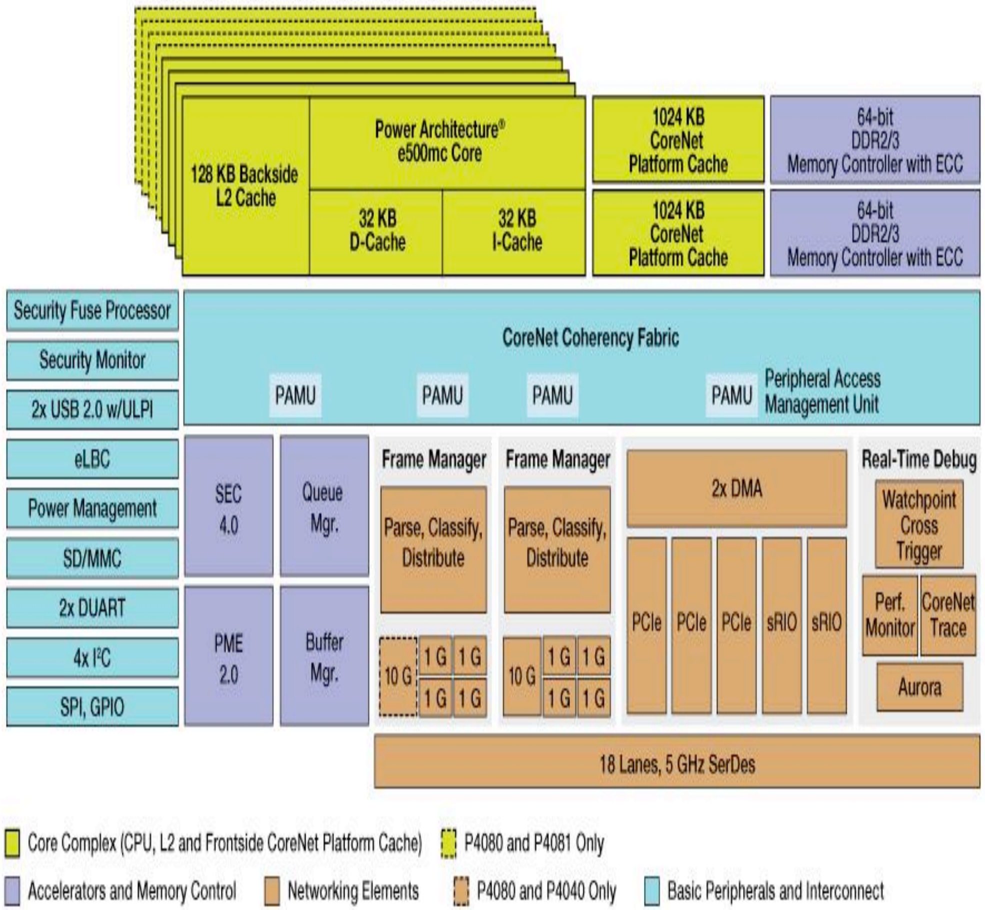




TMDXEVM6678L EVM

- 8 core @ 1.25GHz
- 32 KB L1D and L1P cache.
- 512 KB L2 local cache.
- 4 MB shared L2 cache.
- 8 GB of shared external DDR3 memory at 12.8 GB/s.

Freescale's Communication processor with data path



QorIQ P4080 processor

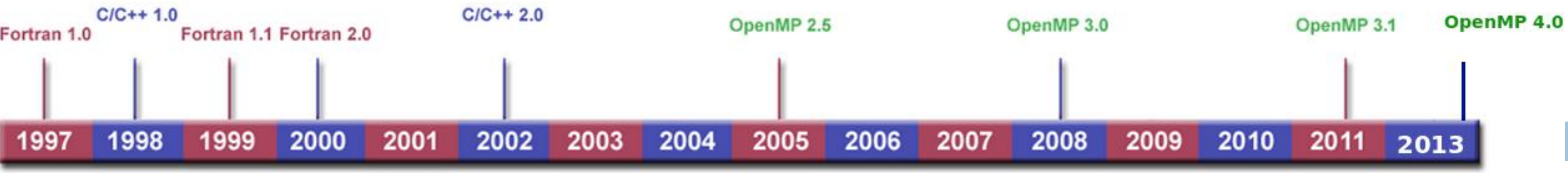
- 4-8 Power architecture e500mc cores
- Accelerators
  - Encryption (SEC)
  - Pattern Matching Engine (PME)
- Target applications:
  - Aerospace and Defense
  - Ethernet Switch, Router
  - Pre-crash detection
  - Forward Collision Warning

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=P4080&tid=redP4040

Programmers' requirements

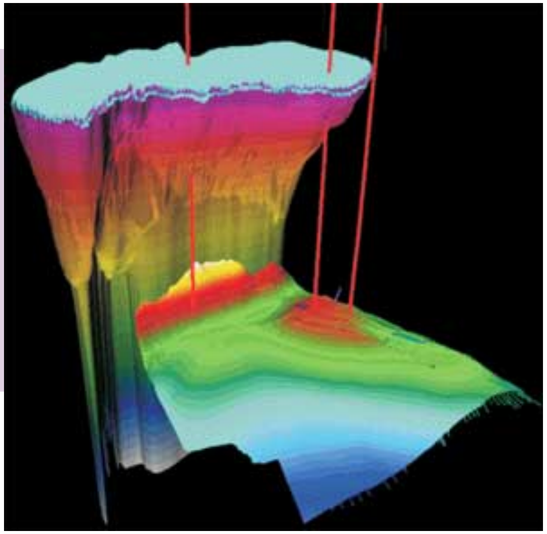
- Rewriting applications from scratch requires considerable time and effort
  - Need easy way to parallelize existing codes
  - Incremental migration path essential for major application codes
  - May need to exploit multiple levels of parallelism
- ...with familiar and/or commodity programming models
  - Not all programming models are created equal
  - None are perfect, but industry adoption is critical

Defacto and mature standard - OpenMP



- High-level API for shared memory programming
  - Widespread vendor support and a large user base
  - User makes strategic decisions; compiler figures out details
- OpenMP code is portable
  - Across compilers, runtimes
  - Mainstream compilers for Fortran, C and C++ support OpenMP

```
#pragma omp parallel
#pragma omp for schedule(dynamic)
     for (I=0;I<N;I++){
          NEAT_STUFF(I);
     } /* implicit barrier here */
```

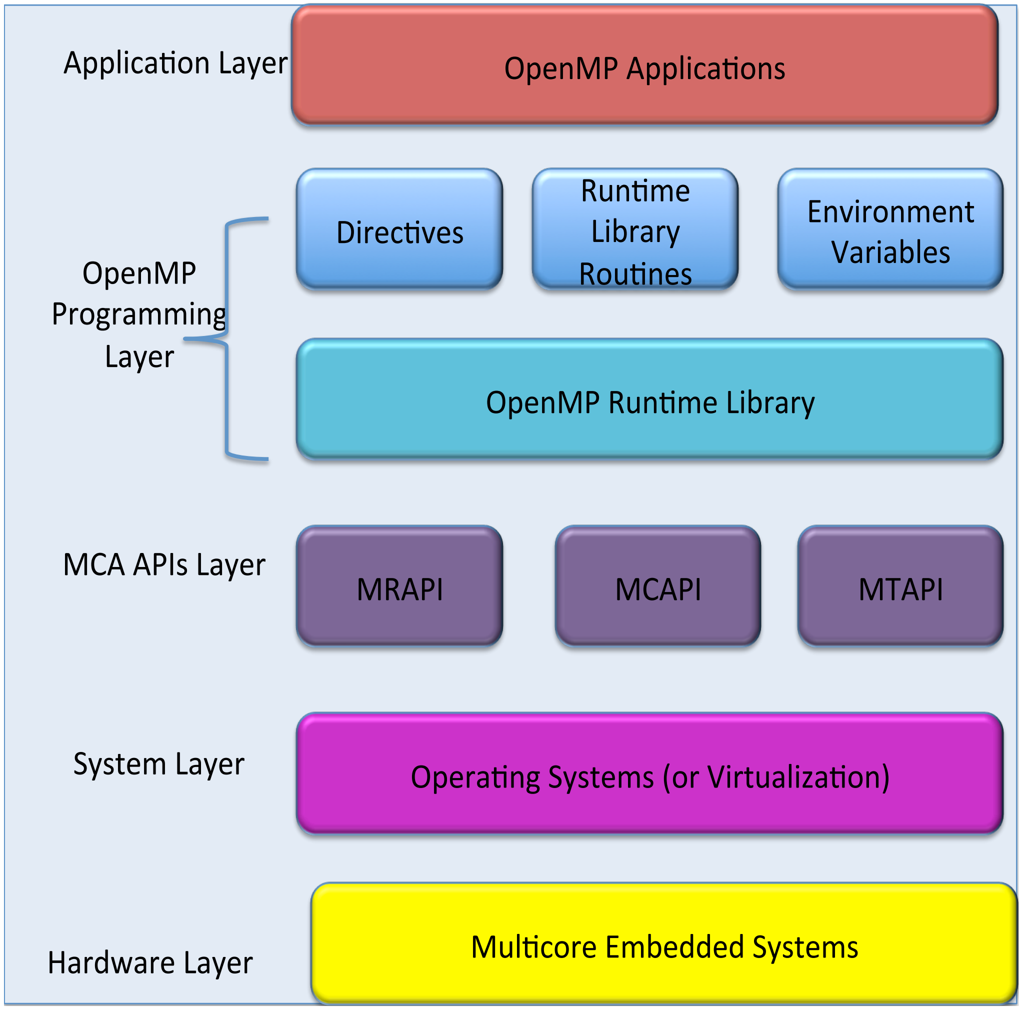


OpenMP for Embedded Systems

- Embedded programmers need portability too
  - Across diverse platforms; supported by multiple compilers and tools
  - Lets programmers focus on the algorithm and not the low-level details of concurrency (v important factor for embedded systems)
- OpenMP seen as very useful in this domain also, but:
  - OpenMP runtime relies on lower level components
    - OS and threading/hardware libraries
    - Memory allocation, synchronization e.g. Linux, Pthreads
      - But embedded systems typically lack some of these features
  - OpenMP has shared-memory cache-coherent memory model
    - However embedded platforms feature distributed, non-uniform memory, with no cache-coherency
- Vocabulary for heterogeneity is required in the embedded space

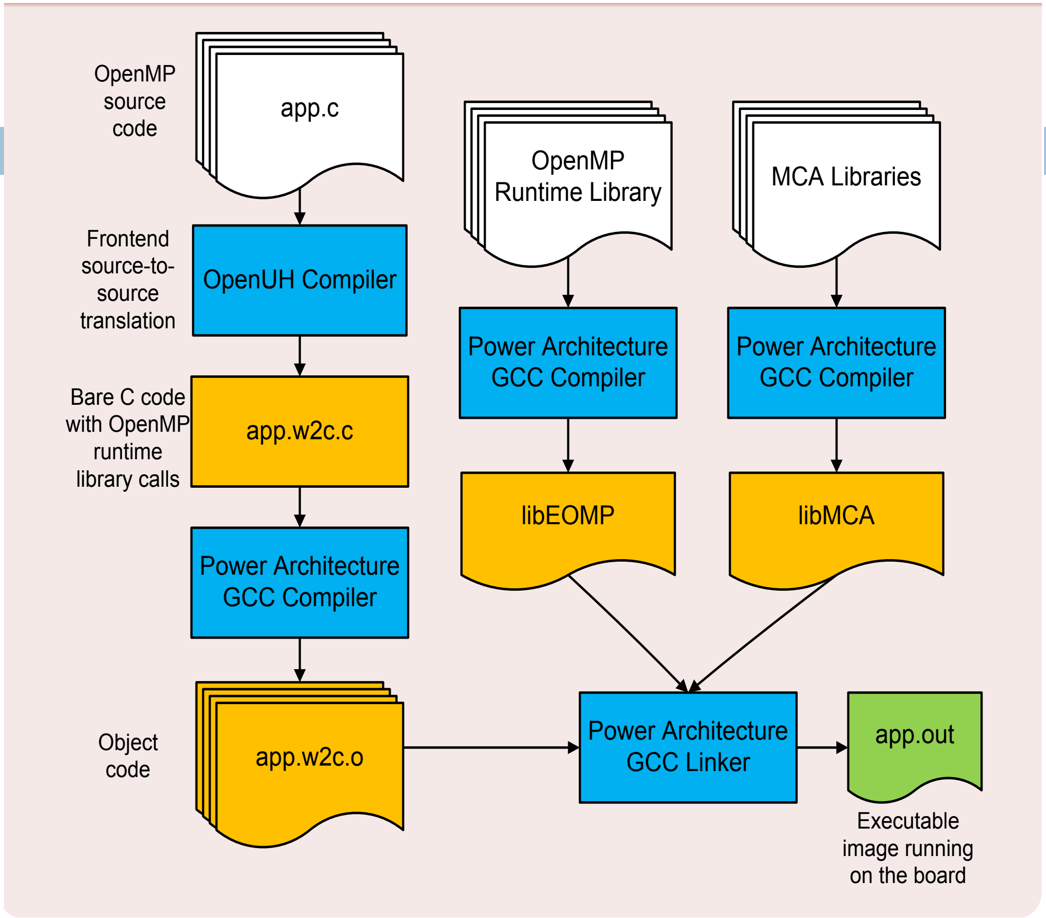
Portable OpenMP Implementation

- Translated OpenMP for MPSoCs
- Used Multicore Association (MCA) APIs as target for our OpenMP translation
- Developed MCA-based runtime:
  - Portable across MPSoCs
  - Light-weight
  - Supports non-cache-coherent systems
  - Performance comparable to customized vendor-specific implementations

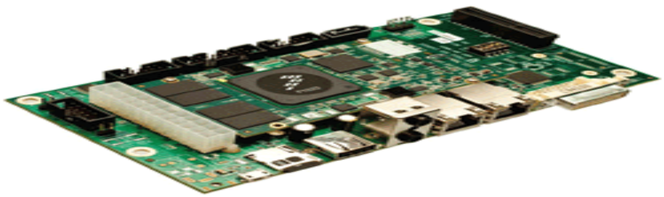


Compilation Process

- OpenUH as our frontend source-to-source compiler
  - Translates C+OpenMP source into C with OpenMP runtime function calls
- PowerPC-GCC as our backend to generate the object file and libraries
- Final executable file is generated by linking the object file, our OpenMP runtime library and the MCA runtime library.



Dual-core power processor from Freescale Semiconductor



Dijkstra Kernel – Case Study


```
    me = omp_get_thread_num();
#pragma omp master
  {  nth = omp_get_num_threads();
     chunk = nv/nth;
  }
  startv = me * chunk;
  endv = startv + chunk - 1;
  for (step = 0; step < nv; step++){
     #pragma omp master
     {  md = largeint; mv = 0;  }
     findmymin(startv,endv,&mymd,&mymv);
     #pragma omp critical
     {  if (mymd < md)    {  md = mymd; mv = mymv;  }
     }
     #pragma omp barrier
     #pragma omp master
     {  notdone[mv] = 0;  }
```

Translation of OpenMP – IR w2c


```
extern void DijkstraKernel()
{
  register _INT32 _w2c___ompv_ok_to_fork;
  register _UINT64 _w2c_reg3;
  register _INT32 _w2c___comma;
  register _INT32 _w2c___comma0;
  _INT32 __ompv_gtid_s1;
  /*Begin_of_nested_PU*/
  _w2c___ompv_ok_to_fork = 1;
  if(_w2c___ompv_ok_to_fork)
  {
    _w2c___ompv_ok_to_fork = __ompc_can_fork();
  }
  if(_w2c___ompv_ok_to_fork)
  {
    __ompc_fork(num_threads, &__omprg_DijkstraKernel_1, _w2c_reg3);
  }
  else
  {
//run the code sequentially
 }
  return;
} /* DijkstraKernel */
```

Calling MCA routines


```
int __ompc_init_rtl()
{
    mca_status_t mrapi_status;
    mrapi_parameters_t parms = 0;
    mrapi_info_t version;
    mrapi_initialize(DOMAIN,NODE,parms,&version,&mrapi_status);
   /* get the number of hardware processors */
    __omp_num_cpus = __omp_get_num_cpu();
    if(__omp_num_cpus == 0){
        init_rtl = -2;
        return init_rtl;
    }
/* number of MCA nodes */
    __omp_nthreads_var = __omp_num_cpus;
/* parse OpenMP environment variables */
    __ompc_parse_environment_variables();
/* register the finalize function*/
    atexit(__ompc_fini_rtl);
/* setup MCA shared memory */
    // ...
    shmem_data = mrapi_shmem_create(OMP_SHMEM_DATA_KEY,__omp_shmem_size ,NULL,0,NULL,0,&mrapi_status);
    // ...

/* setup MCA nodes */
    // ...
    mrapi_initialize(0,node_id,parms,&version,&mrapi_status);
    shmem_data = mrapi_shmem_get(OMP_SHMEM_DATA_KEY,&mrapi_status);
    // ...
    return init_rtl;
}
```

Calling MCA routines


```
void __ompc_fork(const int _num_threads, omp_micro micro_task,frame_pointer_t frame_pointer)
{
    int num_threads = _num_threads;
    int i;
    omp_icv_t* icvs = (omp_icv_t*)addr_data_root;
    omp_mca_node_t* list_nodes = (omp_mca_node_t*)((unsigned long)addr_data_root + icvs->offset_list_nodes);

    mca_status_t mrapi_status;
    mrapi_node_t mrapi_node_id = mrapi_node_id_get(&mrapi_status);
    omp_mca_node_t* n = &(list_nodes[mrapi_node_id]);

    if(n->omp_exe_mode == OMP_EXE_MODE_SEQUENTIAL){
      // ...
/* Master node wake up the child nodes and schedule tasks for the team*/
      // ...

/* barrier */
      __ompc_level_1_barrier(0);
/* back to serial execution */
      n->omp_exe_mode = OMP_EXE_MODE_SEQUENTIAL;
    }
}
```

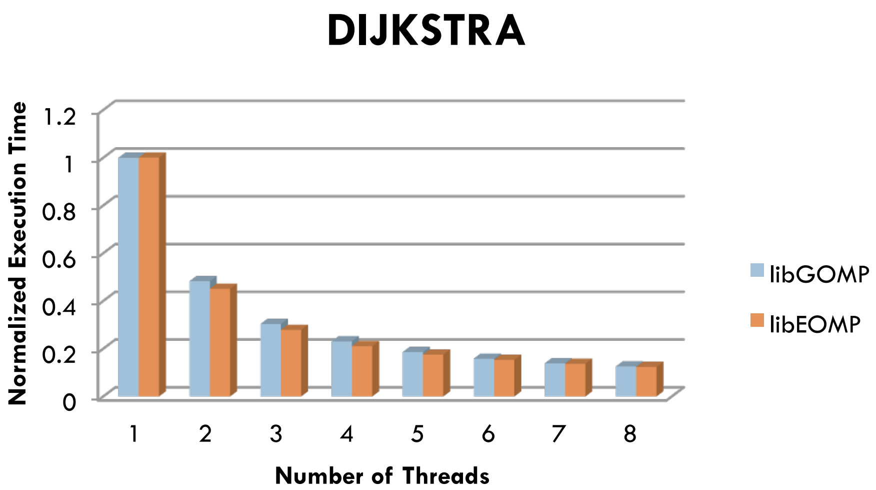
Make file and Compilation output

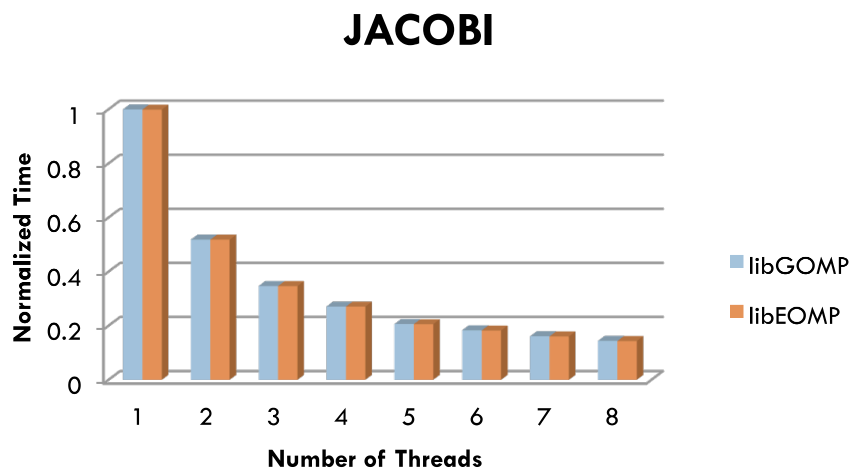

```
a.o Dijkstra_kernel.w2c.o
ra.c Dijkstra_kernel.w2c.c

linux-gnu-gcc
00mc -O3 -std=c99
penmp -lpthread -lm -lmrapi
HOME}/opt/openuh-install/include/4.2
HOME}/mca-build/lib -L ${HOME}/opt/openuh-install/lib/gcc-lib/x86_64-open64-linux/4.2

BJ} Makefile
S} ${OBJ} $(C_INC) $(LDFLAGS) $(C_LIB) -o Dijkstra_mca

ijkstra.c
S} $(C_INC) -c Dijkstra.c
el.w2c.o: Dijkstra_kernel.w2c.c
S} $(C_INC) -c Dijkstra_kernel.w2c.c


: output* *.w2c.c *.w2c.h

Output
-gnu-gcc -te500mc -O3 -std=c99 -w  -I /home/cwang/opt/openuh-install/include/4.2 -c Dijkstra
-gnu-gcc -te500mc -O3 -std=c99 -w  -I /home/cwang/opt/openuh-install/include/4.2 -c Dijkstra
-gnu-gcc -te500mc -O3 -std=c99 -w  Dijkstra.o Dijkstra_kernel.w2c.o
g/opt/openuh-install/include/4.2 -lopenmp -lpthread -lm -lmrapi
g/mca-build/lib -L /home/cwang/opt/openuh-install/lib/gcc-lib/x86_64-open64-linux/4.2 -o Di
```

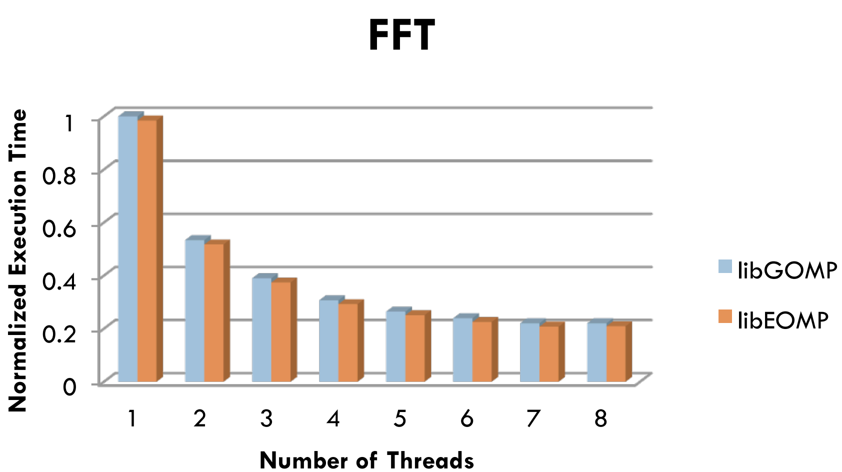
Results

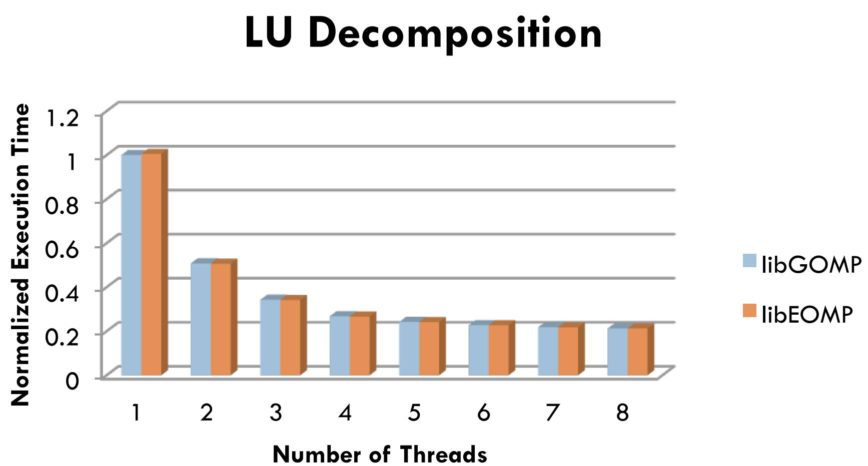

```
#LibEomp with 1 thread
Success to initialize the mrapi
The parallel wall time is 10.461896 sec
deallocating mrapi memory
#LibEomp with 8 threads
Success to initialize the mrapi
The parallel wall time is 1.301495 sec
deallocating mrapi memory
```

Comparison of execution time of our libEOMP with native GCC libGOMP on a Freescale 8-core power processor board









Let’s make programming embedded devices EASY !!

Publications

Cheng Wang, Sunita Chandrasekaran, Barbara Chapman, and Jim Holt. 2013, "libEOMP: a portable OpenMP runtime library based on MCA APIs for embedded systems", in Proceedings of the 2013 International Workshop (PMAM), co-located with 18th ACM SIGPLAN Symposium on (PPoPP), Shenzhen, China, 2013

Cheng Wang, Sunita Chandrasekaran, Peng Sun, Barbara Chapman, and Jim Holt, "Portable Mapping of OpenMP to Multicore Embedded Systems Using MCA APIs", in Proceedings of the 14th ACM SIGPLAN/SIGBED conference on LCTES, pp.153-162, Seattle, WA, June 2013.