Yes, UKAN

Uncommitted **K**AOS **A**ccessible **N**odes

The KAOS (Compilers, Hardware Architectures, and Operating Systems) research group at the University of Kentucky, externally known as **`Aggregate.Org`**, operates multiple cluster supercomputers — and will share free cycles with collaborators and friendly users. The **March 2012** UKAN includes:

Cluster	Nodes	Cores	Mem	Attached
`ack`	8	8	2GB	1GB HD6770
`axk`	16	32	1GB	—
`cik`	20	40	1GB	—
`dax`	18	18	512MB	2x200GB disk
`doc`	24	48	4GB	3x120GB disk
`emc`	22	22	2GB	160GB disk
`hak`	96	96	512MB	—
`nak`	64	64	512MB	1GB 9500GT
`pik`	32	32	512MB	—
`rak`	8	8	1GB	1GB NetFPGA

All clusters run Linux with Gnu tools and OpenMPI; several also support OpenCL. This reference card gives a brief overview of how to use them in UKAN.

Prof. Hank Dietz & “The Aggregate”
Electrical and Computer Engineering Department
University of Kentucky
Lexington, KY 40506-0046
`http://aggregate.org/UKAN/`

Policies for use of UKAN

We are a systems research group happy to freely share our computing resources with worthy users — e.g., potential collaborators or projects that are of general benefit to the community. If you’re developing or improving your own application, we’re happy to help. If you’re looking for production cycles and QoS promises for running a standard program, there’s not much we and UKAN can do for you.

- Although we can treat resources as a sharable pool of nodes, we prefer to allocate an entire cluster to a single parallel program at a time
- We reserve the right to kick jobs off any machine at any time without prior notice
- See **`http://aggregate.org/UKAN/`** for details about requesting access, etc.
- Most cluster head nodes are accessed via **`ssh`** through port 22 of `super.ece.engr.uky.edu`

Using OpenCL/KOAP

The OpenCL standard, **`www.khronos.org/opencl`**, defines an "Open Computing Language" that targets SIMD parallel architectures. OpenCL implements a simple C dialect for writing the SIMD parallel code and provides host library routines to control execution (the OpenCL API). Different implementations target different hardware: Intel and AMD target SSE/AVX SWAR instructions, while NVIDIA and AMD target their respective GPUs.

KOAP (Kentucky OpenCL Application Preprocessor) is a simple tool our group created to hide the complexities of the OpenCL API. It is freely available from **`aggregate.org/KOAP`**.

KOAP Directives

- **`$include`** *`"file"`*
- **`$define`** *`name value`*
- **`$global`**
- **`$init`**
- **`$clalloc(`***`hostname, type, count, perms`***`)`**
- **`$clwrite(`***`hostpointers`***`)`**
- **`$clread(`***`hostpointers`***`)`**
- **`$call`** *`kernel`***`(`***`args, gsize, lsize`***`)`**
- **`$clfree(`***`callocedstuff`***`)`**
- **`$clcleanup`**

OpenCL/KOAP Commands

- To convert KOAP directives:
  `koap` *`file.koap`*
- To compile the OpenCL API program:
  `cc` *`file.c`* `-lOpenCL -o` *`file`*
- To run on whatever (within one node):
  `./file args`

OpenCL/KOAP Sample Program

The following program simply performs an element-by-element add of two random vectors.

```
$define SIMDS 4
$define MAXBLOCK 8
const int NPROC = SIMDS * MAXBLOCK;
const int NBPROC = MAXBLOCK;
$global

${
//pure OpenCL code here...
__kernel void
VecAdd(__global float *a,
       __global float *b,
       __global float *out)
{
 int iproc = get_global_id(0);
 out[iproc] = a[iproc] + b[iproc];
}
$}

main(int argc, char **argv)
{
  float a[NPROC], b[NPROC], out[NPROC];
  int i;
  for (i=0; i<NPROC; ++i)
    { a[i] = rand(); b[i] = rand(); }
  $init
  $clalloc(a, cl_float, NPROC, ro)
  $clalloc(b, cl_float, NPROC, ro)
  $clalloc(out, cl_float, NPROC, wo)
  $clwrite(a, b)
  $call VecAdd(a, b, out, NPROC, NBPROC)
  clFinish(cmdQ);
  $clread(out)
  for (i=0; i<NPROC; ++i)
    printf("%f = %f + %f0,
           out[i], a[i], b[i]);
  $clfree(a, b, out)
  $clcleanup
  return(0);
}
```

Using MPI (Message Passing Interface)

The MPI standard is used to write and run parallel programs across MIMD PEs, be they cores inside a processor or nodes in a cluster — debugging runs do not even require a parallel machine. We generally use OpenMPI, **www.open-mpi.org**, with the C programming language.

MPI Environment

- **#include <mpi.h>**
- int **MPI_Init**(int **argc, char ***argv)
- int **MPI_Finalize**(void)

MPI Common Constants

- **MPI_SUCCESS**, **MPI_ERR**, **MPI_ANY_TAG**, **MPI_ANY_SOURCE**, **MPI_COMM_WORLD**, **MPI_NULL_WINDOW**
- **MPI_Datatype** values: **MPI_BYTE**, **MPI_CHAR**, **MPI_SHORT**, **MPI_INT**, **MPI_LONG**, **MPI_UNSIGNED_CHAR**, **MPI_UNSIGNED_SHORT**, **MPI_UNSIGNED**, **MPI_UNSIGNED_LONG**, **MPI_FLOAT**, **MPI_DOUBLE**
- **MPI_Op** values: **MPI_MAX**, **MPI_MIN**, **MPI_SUM**, **MPI_PROD**, **MPI_BAND**, **MPI_BOR**, **MPI_BXOR**, **MPI_LAND**, **MPI_LOR**, **MPI_LXOR**, **MPI_REPLACE**

MPI Message Passing

- int **MPI_Send**(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm c)
- int **MPI_Recv**(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm c, MPI_Status *s)
- int **MPI_Sendrecv**(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int src, int recvtag, MPI_Comm c, MPI_Status *s)
- int **MPI_Sendrecv_replace**(void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int src, int recvtag, MPI_Comm c, MPI_Status *s)
- int **MPI_Get_count**(MPI_Status *s, MPI_Datatype datatype, int *count)
- int **MPI_Iprobe**(int src, int tag, MPI_Comm c, int *flag, MPI_Status *s)

MPI Collective Communications

- int **MPI_Bcast**(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm c)
- int **MPI_Allgather**(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm c)
- int **MPI_Gather**(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm c)
- int **MPI_Scatter**(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm c)
- int **MPI_Alltoall**(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm c)
- int **MPI_Reduce**(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm c)
- int **MPI_Allreduce**(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm c)
- int **MPI_Scan**(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm c)
- int **MPI_Barrier**(MPI_Comm c)

MPI Remote Memory Access

- int **MPI_Win_create**(void *base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm c, MPI_Win *w)
- int **MPI_Win_free**(MPI_Win *w)
- int **MPI_Put**(void *orgaddr, int orgcount, MPI_Datatype orgtyp, int targrank, MPI_Aint targdisp, int targcount, MPI_Datatype targtyp, MPI_Win w)
- int **MPI_Get**(void *orgaddr, int orgcount, MPI_Datatype orgtyp, int targrank, MPI_Aint targdisp, int targcount, MPI_Datatype targtyp, MPI_Win w)
- int **MPI_Accumulate**(void *orgaddr, int orgcount, MPI_Datatype orgtyp, int targrank, MPI_Aint targdisp, int targcount, MPI_Datatype targtyp, MPI_Op op, MPI_Win w)
- int **MPI_Win_fence**(int assert, MPI_Win w)

MPI Commands

- To compile:
  mpicc *file.c* -o *file*
- To run on *N* PEs in nodes listed in *names*:
  mpirun -n *N* -hostfile *names file*

MPI Sample Program

Compute approximate value of π; algorithm from M. J. Quinn, *Parallel Computing Theory And Practice*, McGraw Hill, 1994.

```
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
{
  register double width;
  double sum, lsum;
  double rsum = 0.0;
  register int intervals, i;
  int nproc, iproc;

  if (MPI_Init(&argc, &argv) !=
      MPI_SUCCESS) exit(1);
  MPI_Comm_size(MPI_COMM_WORLD, &nproc);
  MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
  intervals = atoi(argv[1]);
  width = 1.0 / intervals;
  lsum = 0;
  for (i=iproc; i<intervals; i+=nproc) {
    double x = (i + 0.5) * width;
    lsum += 4.0 / (1.0 + x * x);
  }
  lsum *= width;
  MPI_Reduce(&lsum, &sum, 1, MPI_DOUBLE,
             MPI_SUM, 0, MPI_COMM_WORLD);
  if (iproc == 0)
    printf("Pi=%f or %f\n", sum, rsum);
  MPI_Finalize();
  return(0);
}
```