UPC Random Access Designed for Speed

Nathan Wichmann

Cray Inc.

wichmann@cray.com

UPC Random Access: Designed for Speed

- This version of UPC Random Access was originally written in Spring 2004
- Written to maximize speed
- Had to work inside of the HPCC benchmark
- Had to run well on any number of CPUs
- Also happens to be a very productive way of writing the Global RA.

UPC Random Access: Highlights

- Trivial to parallelize, each PE gets its share of updates
  - Strip-mine loop to expose loop level parallelism to compiler
- Unified Parallel C allows direct, one-sided access to distributed variables; NO two-sided messages!
- Decomposed “Table” into 2 Dims. to allow explicit, fast computation of LocalOffset and PE number
- Cast integer variables to doubles to do a much faster “integer divide”

Productivity: Fewer lines of code

UPC VERSION

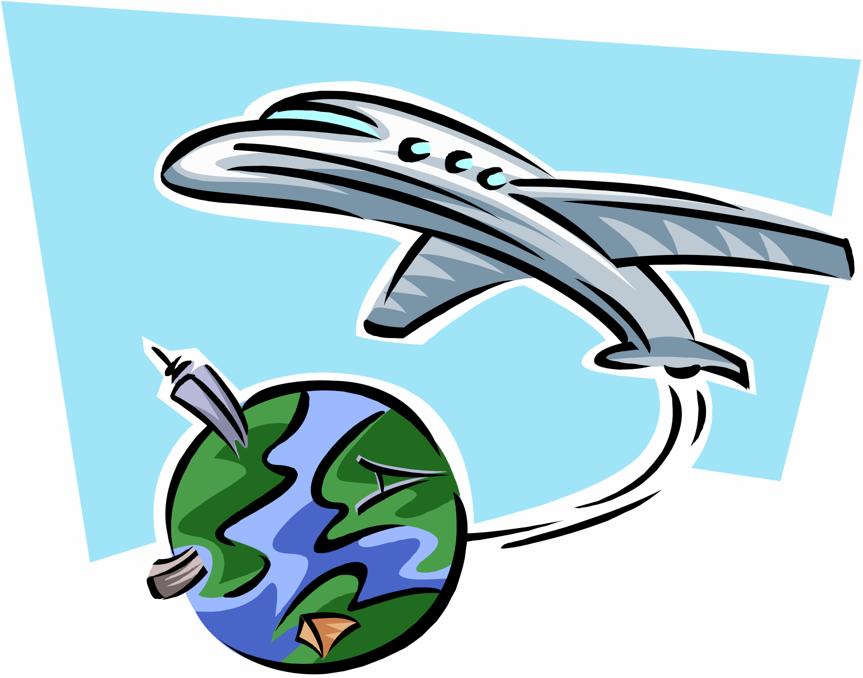
```
#pragma _CRI concurrent
for (j=0; j<STRIPSIZE; j++)
 for (i=0; i<SendCnt/STRIPSIZE; i++) {
  VRan[j] = (VRan[j] << 1) ^ ((s64Int) VRan[j]<
    ZERO64B ? POLY : ZERO64B);
  GlobalOffset = VRan[j] & (TableSize - 1);
  if (PowerofTwo)
    LocalOffset=GlobalOffset>>logNumProcs ;
  else
    LocalOffset=(double)GlobalOffset/(double)TH
    READS;
    WhichPe=GlobalOffset-LocalOffset*THREADS;
    Table[LocalOffset][WhichPe] ^= VRan[j] ;
 }
}
```

BASE VERSION

```
NumRecvs = (NumProcs > 4) ?(Mmin(4,MAX_RECV))
    : 1;
  for (j = 0; j < NumRecvs; j++)
    MPI_Irecv(&LocalRecvBuffer[j*LOCAL_BUFFER
    _SIZE], localBufferSize,INT64_DT,
    MPI_ANY_SOURCE, MPI_ANY_TAG,
    MPI_COMM_WORLD,&inreq[j]);
while (i < SendCnt) {
do {
MPI_Testany(NumRecvs, inreq, &index,
    &have_done, &status);
if (have_done) {
 if (status.MPI_TAG == UPDATE_TAG) {
   MPI_Get_count(&status, INT64_DT,
    &recvUpdates);
bufferBase = index*LOCAL_BUFFER_SIZE;
for (j=0; j < recvUpdates; j ++) {
 inmsg = LocalRecvBuffer[bufferBase+j];
 LocalOffset = (inmsg & (TableSize - 1)) -
    GlobalStartMyProc;
 HPCC_Table[LocalOffset] ^= inmsg;
 }
 } else if (status.MPI_TAG == FINISHED_TAG) {
    NumberReceiving--;
 } else {
    abort();
 }
```

Productivity : Fewer lines of code

UPC VERSION

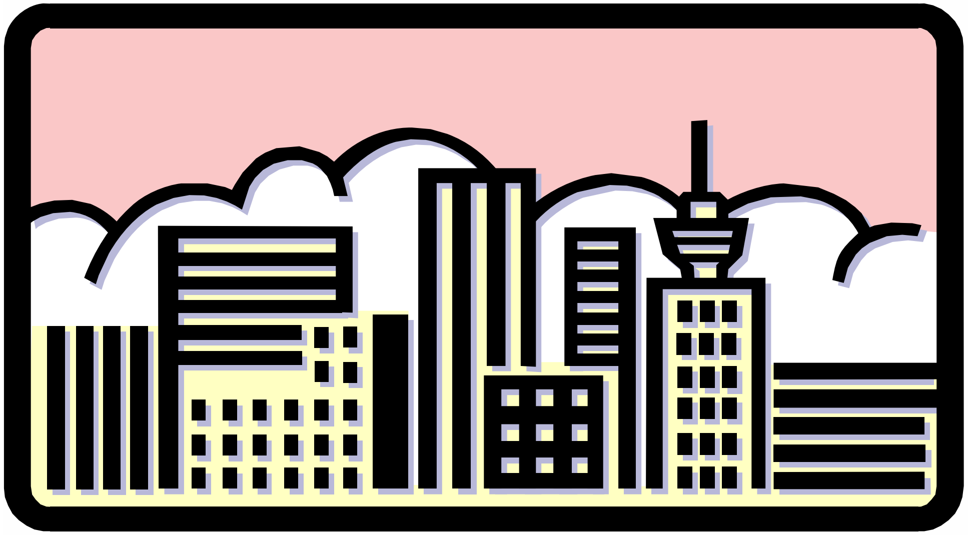


BASE VERSION

```
MPI_Irecv(&LocalRecvBuffer[index*LOCAL_BUFFER
    _SIZE], localBufferSize,INT64_DT,
    MPI_ANY_SOURCE, MPI_ANY_TAG,
    MPI_COMM_WORLD,&inreq[index]);
}
} while (have_done && NumberReceiving > 0);
  if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ ((s64Int) Ran <
    ZERO64B ? POLY : ZERO64B);
    GlobalOffset = Ran & (TableSize-1);
    if ( GlobalOffset < Top)
      WhichPe = ( GlobalOffset /
    (MinLocalTableSize + 1) );
   else
    WhichPe = ( (GlobalOffset - Remainder) /
    MinLocalTableSize );
   if (WhichPe == MyProc) {
    LocalOffset = (Ran & (TableSize - 1)) -
    GlobalStartMyProc;
    HPCC_Table[LocalOffset] ^= Ran;
   }
   else {
    HPCC_InsertUpdate(Ran, WhichPe, Buckets);
        pendingUpdates++;
   }
   i++;
 }
 else {
```

Productivity : Fewer lines of code

UPC VERSION



BASE VERSION

```
MPI_Test(&outreq, &have_done,
    MPI_STATUS_IGNORE);
   if (have_done) {
    outreq = MPI_REQUEST_NULL;
    pe = HPCC_GetUpdates(Buckets,
    LocalSendBuffer, localBufferSize,
    &peUpdates);
    MPI_Isend(&LocalSendBuffer, peUpdates,
    INT64_DT, (int)pe, UPDATE_TAG,
    MPI_COMM_WORLD, &outreq);
    pendingUpdates -= peUpdates;
   }}}
while (pendingUpdates > 0) {
do {
MPI_Testany(NumRecvs, inreq, &index,
    &have_done, &status);
if (have_done) {
  if (status.MPI_TAG == UPDATE_TAG) {
   MPI_Get_count(&status, INT64_DT,
    &recvUpdates);
   bufferBase = index*LOCAL_BUFFER_SIZE;
  for (j=0; j < recvUpdates; j ++) {
   inmsg = LocalRecvBuffer[bufferBase+j];
   LocalOffset = (inmsg & (TableSize - 1)) -
    GlobalStartMyProc;
   HPCC_Table[LocalOffset] ^= inmsg;
   }
} else if (status.MPI_TAG == FINISHED_TAG) {
  NumberReceiving--;
```

Productivity : Fewer lines of code

UPC VERSION

BASE VERSION

```
} else {
  abort();}
MPI_Irecv(&LocalRecvBuffer[index*LOCAL_BUFFER_SI
    ZE], localBufferSize,INT64_DT,
    MPI_ANY_SOURCE, MPI_ANY_TAG,
    MPI_COMM_WORLD,&inreq[index]);
}} while (have_done && NumberReceiving > 0);
    MPI_Test(&outreq, &have_done,
    MPI_STATUS_IGNORE);
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets,
    LocalSendBuffer, localBufferSize,
    &peUpdates);
    MPI_Isend(&LocalSendBuffer, peUpdates,
    INT64_DT, (int)pe, UPDATE_TAG,
    MPI_COMM_WORLD, &outreq);
    pendingUpdates -= peUpdates;
    } }
for (proc_count = 0 ; proc_count < NumProcs ;
    ++proc_count) {
  if (proc_count == MyProc) { finish_req[MyProc]
    = MPI_REQUEST_NULL; continue; }
  MPI_Isend(&Ran, 1, INT64_DT, proc_count,
    FINISHED_TAG,MPI_COMM_WORLD, finish_req +
    proc_count);
  }
while (NumberReceiving > 0) {
```

Productivity : Fewer lines of code

UPC VERSION

BASE VERSION

```
MPI_Waitany(NumRecvs, inreq, &index,
    &status);
if (status.MPI_TAG == UPDATE_TAG) {
  MPI_Get_count(&status, INT64_DT,
    &recvUpdates);
  bufferBase = index * LOCAL_BUFFER_SIZE;
for (j=0; j < recvUpdates; j ++) {
  inmsg = LocalRecvBuffer[bufferBase+j];
  LocalOffset = (inmsg & (TableSize - 1)) -
    GlobalStartMyProc;
  HPCC_Table[LocalOffset] ^= inmsg;
  }
  } else if (status.MPI_TAG == FINISHED_TAG){
     NumberReceiving--;
    } else {
      abort(); }
MPI_Irecv(&LocalRecvBuffer[index*LOCAL_BUFFER
    _SIZE], localBufferSize,INT64_DT,
    MPI_ANY_SOURCE, MPI_ANY_TAG,
    MPI_COMM_WORLD, &inreq[index]);
}
MPI_Waitall( NumProcs, finish_req,
    finish_statuses);
HPCC_FreeBuckets(Buckets, NumProcs);
for (j = 0; j < NumRecvs; j++) {
    MPI_Cancel(&inreq[j]);
    MPI_Wait(&inreq[j], &ignoredStatus);
  }
```

Productivity: Algorithm Transparency

```
#pragma _CRI concurrent
for (j=0; j<STRIPSIZE; j++)
 for (i=0; i<SendCnt/STRIPSIZE; i++) {
  VRan[j] = (VRan[j] << 1) ^ ((s64Int)VRan[j]
           < ZERO64B ? POLY : ZERO64B);
  GlobalOffset = VRan[j] & (TableSize - 1);

  if (PowerofTwo)
    LocalOffset=GlobalOffset>>logNumProcs ;
  else
    LocalOffset=
       (double)GlobalOffset/(double)THREADS;
  WhichPe=GlobalOffset-LocalOffset*THREADS;

  Table[LocalOffset][WhichPe] ^= VRan[j] ;
}}
```

Generate Random Number

Compute GO

Decompose GO into LO and WhichPE

XOR VRan and Table

SPEED!

Tell the compiler the following loop is parallel

```
#pragma _CRI concurrent
```

STRIP-MINE to create loop level parallelism

```
for (j=0; j<STRIPSIZE; j++)
 for (i=0; i<SendCnt/STRIPSIZE; i++) {
  VRan[j] = (VRan[j]<<1) ^ ((s64Int)VRan[j]<
            ZERO64B ? POLY : ZERO64B);
  GlobalOffset = VRan[j] & (TableSize - 1);
```

INVARIANT IF!!

```
  if (PowerofTwo)
```

If PowerofTwo CPUS, LO calc is simple shift

```
    LocalOffset=GlobalOffset>>logNumProcs ;
  else
```

Else, cast variables to double and do a FP divide; MUCH FASTER!!

```
    LocalOffset=(double)GlobalOffset/(double)THREADS;
```

Calculate on WhichPE the Table location resides

```
  WhichPe=GlobalOffset-LocalOffset*THREADS;
  Table[LocalOffset][WhichPe] ^= VRan[j] ;}}
```

Simply access and update a 2D Table

Productivity + Speed = Results

- UPC Random Access sustains 7.69 GUPs on 1008 Cray X1E MSPs.
- Works inside the HPCC framework
- Is "in the spirit" of the benchmark
- Easy to understand and modify if computations are more complex

- The Future
  - Atomic XORs will vastly improve performance
    - All memory references will be "Fire and Forget"

Acknowledgements

- Oak Ridge National Laboratory for a me allowing me to use their machine
- Engineers at Cray for designing and building a really nice machine!
- The compiler writers at Cray for making my code run fast
- All of the people inside and outside of Cray whom I have had interactions with about this project
- Cray Inc for paying me to do this