PARTNERSHIP
FOR ADVANCED COMPUTING
IN EUROPE

HMPP port

G. Colin de Verdière (CEA)

Overview

.Uchu prototype

HMPP

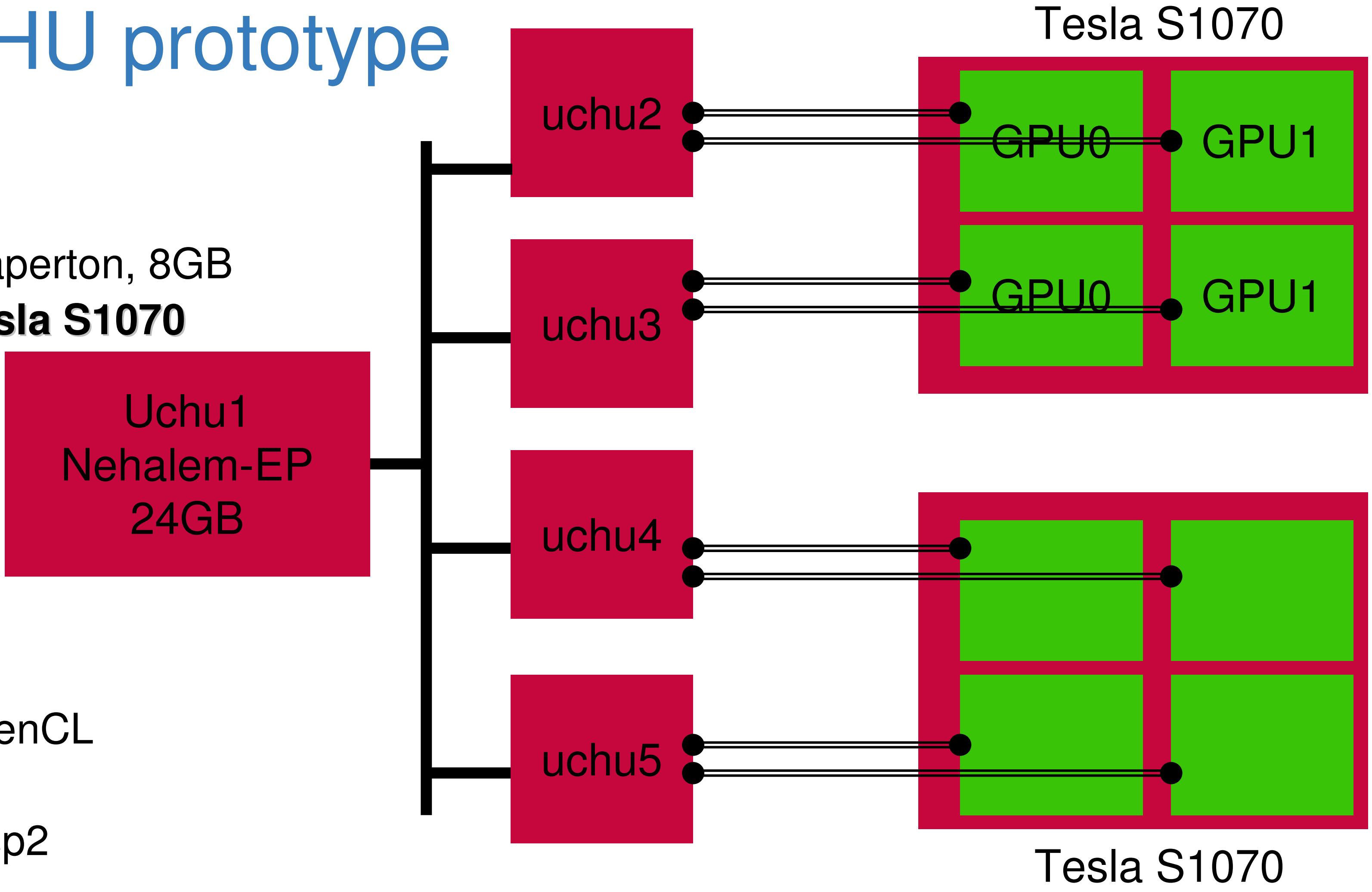
MOD2AS

MOD2AM

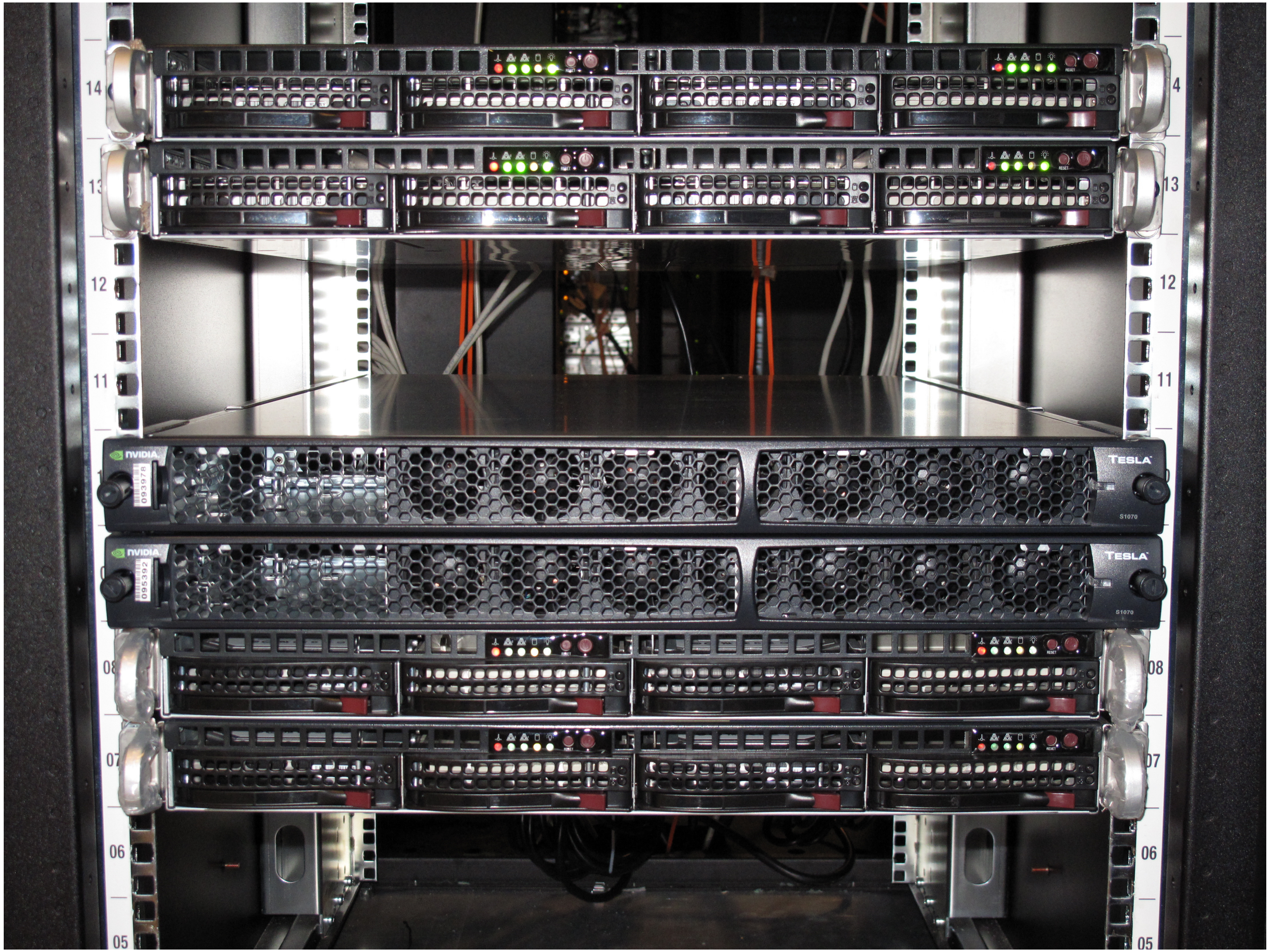
HMPP in a real code

The UCHU prototype

- Bull servers
  - 1 login node
  - 4 nodes 2 Haperton, 8GB
  - 2 **NVIDIA Tesla S1070**
  - IB DDR
- Slurm
- Software
  - gcc/icc/ifort
  - Cuda 2.3/OpenCL
  - Rapidmind
  - HMPP 2.2.1sp2



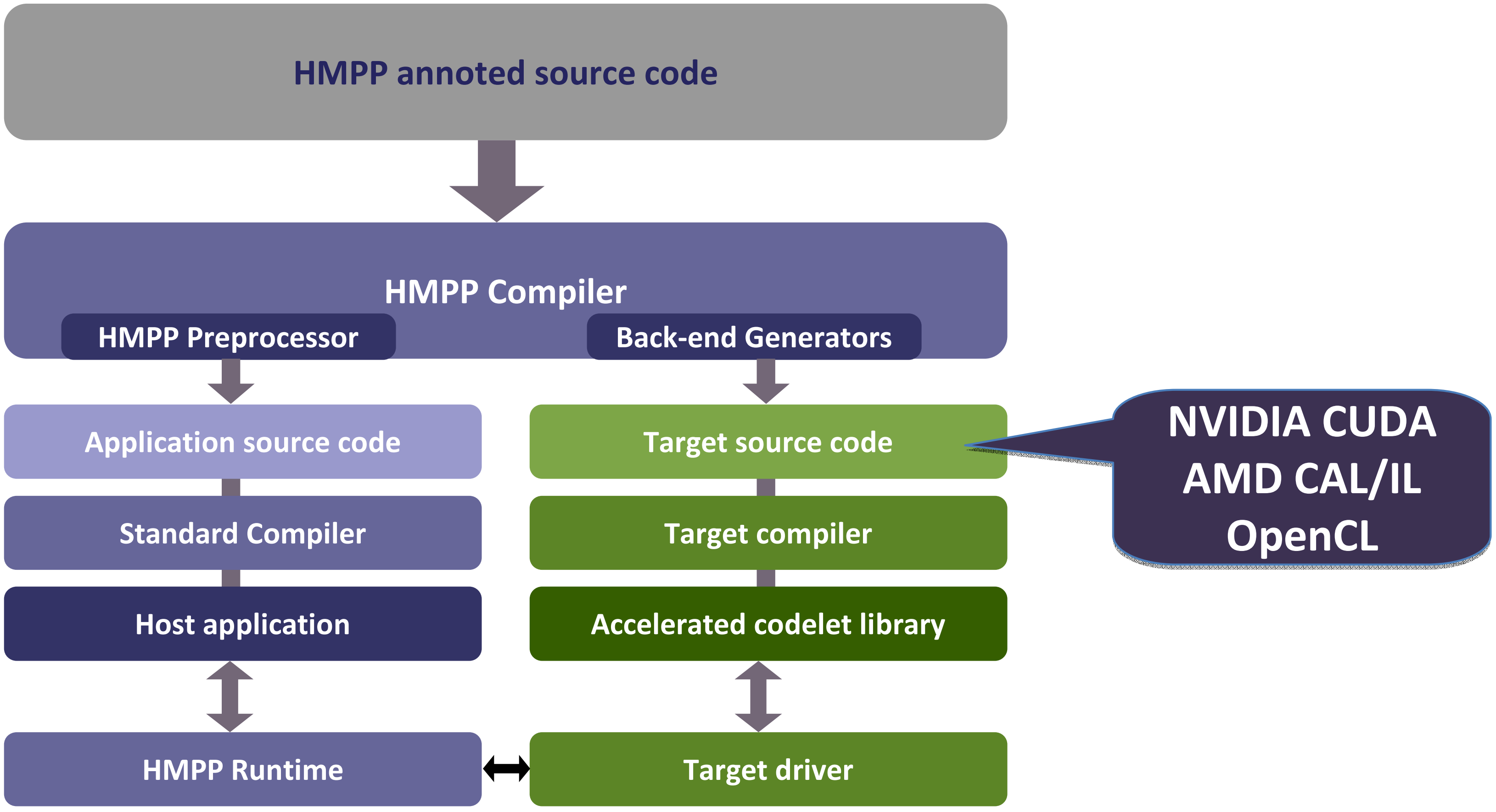
The real UCHU



HMPP

- HMPP is a high-level abstraction for manycore programming
- Takes annotated source code (pragma)
  - Keeps code portability
    - C & Fortran
- Has a low learning curve yet can reach significant performances
- Offers many high level features
  - Asynchronous data transfers and kernel execution
- Offers target oriented optimisations

HMPP: Compilation workflow



CPU GPU

HMPP: a small example

```
#pragma hmpp sgemm codelet, target=CUDA, args[vin1,vin2,vout].io=inout
void sgemm( int m, int n, int k, float alpha,
            float vin1[n][n], float vin2[n][n],
            float beta, float vout[n][n] ) {
 /* . . . */
}

int main(int argc, char **argv) {
    /* . . . */

 for( j = 0 ; j < NB_RUNS ; j++ ) {
#pragma hmpp sgemm callsite
    sgemm( size, size, size, alpha, vin1, vin2, beta, vout );
  }
    /* . . . */
 }
```

Declare codelet with CUDA target

Synchronous codelet call

HMPP: a more elaborate version

Preload data

```
int main(int argc, char **argv) {
    /* . . . */
#pragma hmpp sgemm allocate
    /* . . . */
#pragma hmpp sgemm advancedload, args[vin1;m;n;k;alpha;beta]
    /* . . . */

 for( j = 0 ; j < 2 ; j++ ) {
#pragma hmpp sgemm callsite &
#pragma hmpp sgemm  args[m;n;k;alpha;beta;vin1].advancedload=true
    sgemm( size, size, size, alpha, vin1, vin2, beta, vout );
    /* . . . */
 }

    /* . . . */
#pragma hmpp sgemm release
```

Avoid reloading data

MOD2AM: original code

```
void
mxm(int m, int l, int n, double
    a[m][l], double b[l][n], double
    c[m][n])
{
   int i, j, k, lf, mf;

   mf = m - m % 4;

   for (i = 0; i < mf; i += 4) {
      for (j = 0; j < n; j++) {
         c[i][j] = 0.0;
         c[i + 1][j] = 0.0;
         c[i + 2][j] = 0.0;
         c[i + 3][j] = 0.0;
      }   }
```

```
   lf = l - l % 4;
   for (i = 0; i < m; i++) {
      for (j = 0; j < lf; j += 4) {
         for (k = 0; k < n; k++) {
            c[i][k] = c[i][k]
               + a[i][j + 0] * b[j + 0][k]
               + a[i][j + 1] * b[j + 1][k]
               + a[i][j + 2] * b[j + 2][k]
               + a[i][j + 3] * b[j + 3][k];
         }     }    }

   for (i = 0; i < m; i++) {
      for (j = lf; j < l; j++) {
         for (k = 0; k < n; k++) {
               c[i][k] = c[i][k] + a[i][j] *
   b[j][k];
      }    }    }
} // mxm
```

MOD2AM: HMPP version – main()

```
#pragma hmpp Hmxm advancedload, args[a;b], args[a].size={m,l},
   args[b].size={l,n}
for (i = 0; i < nrep; i++) {
#pragma hmpp Hmxm callsite, args[a;b].advancedload=true
   mxm(m, l, n, (double (*)[m]) a, (double (*)[n]) b, (double
   (*)[n]) c);
}
#pragma hmpp Hmxm delegatedstore, args[c]
time = cclock() - time;
ok = check(m, l, n, (double (*)[n]) c);
```

MOD2AM: mxm.h

```
#pragma hmpp Hmxm codelet,
  args[a;b].io=in, args[c].io=out,
  TARGET=CUDA, args[a].size={m,l},
  args[b].size={l,n}, args[c].size={m,n}

void mxm(int m, int l, int n, double
  a[m][l], double b[l][n], double
  c[m][n]);
```

MOD2AM: mxm.c optimized

```
#include ‘’ mxm.h ‘’
void mxm (int m, int l, int n, const double
   a[m][l], const double b[l][n],
double c[m][n])
{
   #define HUNROLL 2
   int i, j, k, lf, mf;
   int rest = n % HUNROLL;
   for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
                c[i][j] = 0.0;
        }
   }
```

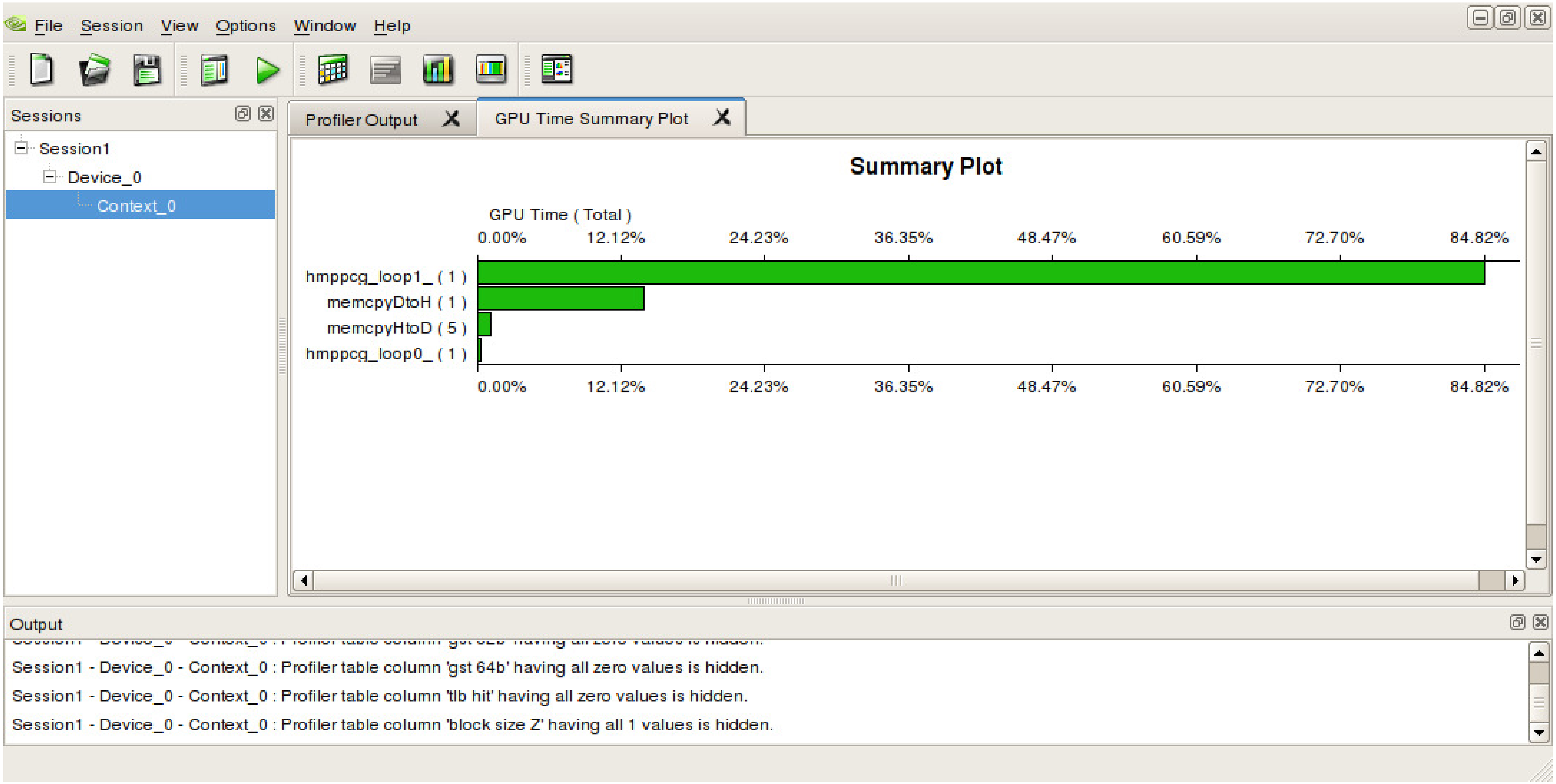
```
#pragma hmppcg unroll(6), jam(k)
for (i = 0; i < m; i++) {

#pragma hmppcg unroll(2), split, noremainder
   for (k = 0; k < n - rest; k++) {
        double prod = 0.0;
        double va, vb;
        j = 0;
        va = a[i][j]; // trick: prefetching of arrays a and b
        vb = b[j][k];
        for (j = 1; j < l; j++) { // prefetching: index 0 has been loaded
                prod += va * vb;
                va = a[i][j];
                vb = b[j][k];
        }
        prod += va * vb; // prefetching: don’t forget the last index
        c[i][k] += prod;
   }
}
```

```
  if (rest) {      // leftovers using the same trick
      for (i = 0; i < m; i++) {
            for (k = n - rest; k < n; k++) {
                   double prod = 0.0;
                   double va, vb;
                   j = 0;
                   va = a[i][j];
                   vb = b[j][k];
                   for (j = 1; j < l; j++) {
                          prod += va * vb;
                          va = a[i][j];
                          vb = b[j][k];
                   }
                   prod += va * vb;
                   c[i][k] += prod;
      }}
  }
}
```

MOD2AM: results on uchu (78GF DP peak)

```
env HMPPCG_FLAGS="--cuda-block-size 64x1" ccc_mprun ./h.mod2am
    10 |   10 |    10|     0.0317 |       0.06 |  T |
    20 |   20 |    20|     0.0021 |       7.62 |  T |
    50 |   50 |    50|     0.0052 |      48.18 |  T |
   100 |  100 |   100|     0.0103 |     193.81 |  T |
   192 |  192 |   192|     0.0173 |     817.62 |  T |
   200 |  200 |   200|     0.0262 |     610.66 |  T |
   500 |  500 |   500|     0.0609 |    4107.86 |  T |
   512 |  512 |   512|     0.0639 |    4201.46 |  T |
   576 |  576 |   576|     0.0642 |    5954.20 |  T |
  1000 | 1000 |  1000|     0.1221 |   16381.76 |  T |
  1024 | 1024 |  1024|     0.1254 |   17129.71 |  T |
  2000 | 2000 |  2000|     0.4590 |   34854.74 |  T |
  2048 | 2048 |  2048|     0.5092 |   33741.00 |  T |
  5000 | 5000 |  5000|     6.5653 |   38078.93 |  T |
 10240 |10240 | 10240|    53.3472 |   40254.82 |  T |
 12096 |  576 | 12096|     4.7198 |   35711.70 |  T |
 12096 | 1152 | 12096|     8.2283 |   40968.88 |  T |
```



```
Session1 - Device_0 - Context_0 : Profiler table column 'gst 64b' having all zero values is hidden.
Session1 - Device_0 - Context_0 : Profiler table column 'tlb hit' having all zero values is hidden.
Session1 - Device_0 - Context_0 : Profiler table column 'block size Z' having all 1 values is hidden.
```

G. Colin de Verdière, CEA

MOD2AS: original code

```
#endif
void
SPMXV(int ncols, int nrows, int nelmts, int indx[nelmts], int rowp[nrows],
REAL matvals[nelmts], REAL invec[ncols], REAL outvec[nrows])
{
int i, j, nrest;
for (i = 0; i < nrows; i++)
        outvec[i] = (REAL) 0.0;
for (i = 0; i < nrows - 1; i++) {
        nrest = (rowp[i + 1] - rowp[i]) % 4;
        for (j = rowp[i]; j < rowp[i + 1] - nrest; j += 4) {
                outvec[i] = outvec[i] + (matvals[j] * invec[indx[j]])
                        + matvals[j + 1] * invec[indx[j + 1]]
                        + matvals[j + 2] * invec[indx[j + 2]]
                        + matvals[j + 3] * invec[indx[j + 3]];
        }
        for (j = rowp[i + 1] - nrest; j < rowp[i + 1]; j++) {
                outvec[i] = outvec[i] + matvals[j] * invec[indx[j]];
        }
}
for (j = rowp[nrows - 1]; j < nelmts; j++) {
        outvec[nrows - 1] = outvec[nrows - 1] + matvals[j] * invec[indx[j]];
}
}
```

MOD2AS: main()

```
#define LOOPS 1
for (j = 0; j < LOOPS; j++) {
   for (i = 0; i < nrows; i++)
        outvec[i] = (REAL) 0.0;
#pragma hmpp Hspmxv callsite
   spmxv(ncols, nrows, nelmts, indx, rowp, matvals,
   invec, outvec);
   time = cclock() - time;
}
time /= LOOPS;
ok = check(ncols, nrows, nelmts, indx, rowp, outvec);
```

MOD2AS: spmxv.c

```
void
spmxv(int ncols, int nrows, int nelmts, int
   indx[nelmts], int rowp[nrows],
double matvals[nelmts], double invec[ncols], double
   outvec[nrows])
{
int i, j, k;
int start, end;
int nrest = 0;
int warpoff = 0;
double somme = 0.;
#define SIZE 32*2
double partial[nrows * SIZE];
for (i = 0; i < nrows; i++) {
   outvec[i] = (double) 0.0;
}
```

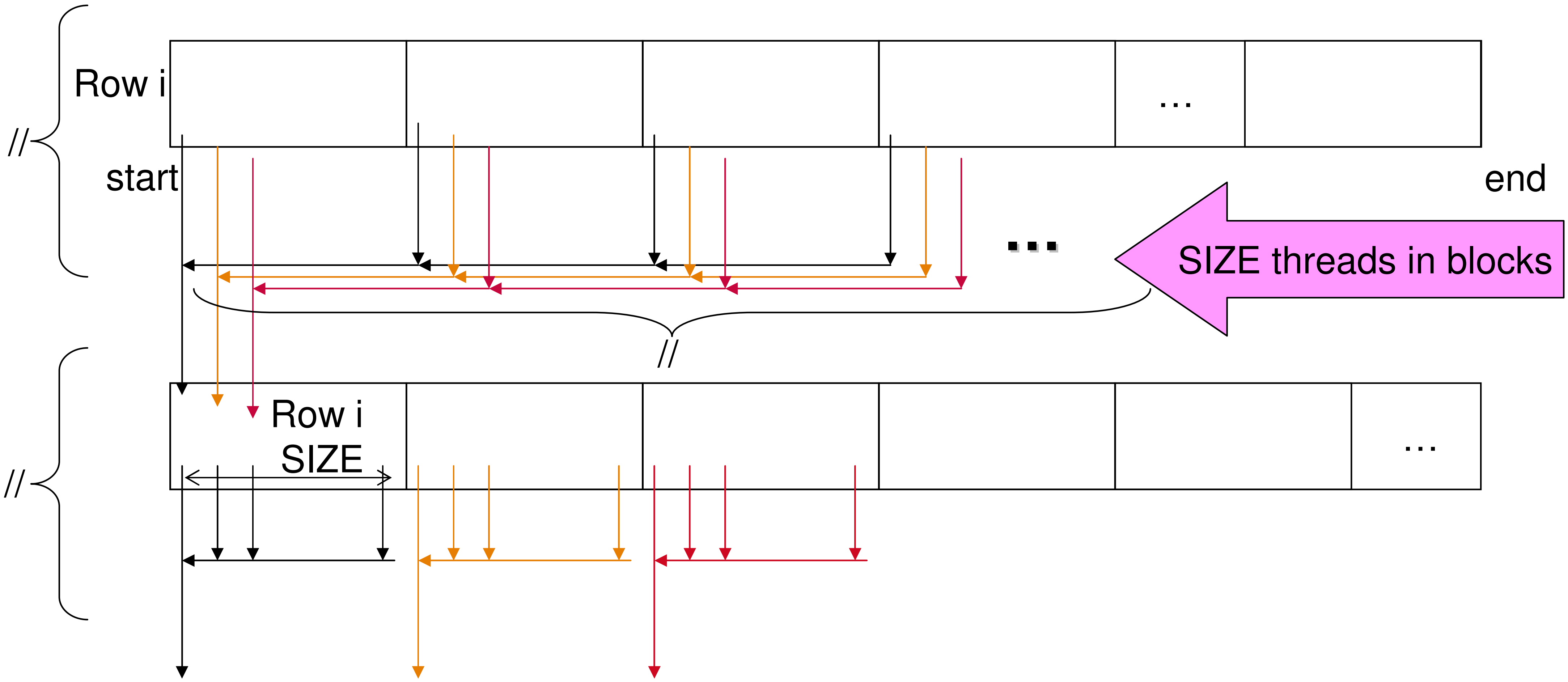
2D
Grid

```
// do partial sums in // taking warp sizes into account
for (i = 0; i < nrows; i++) {
  for (warpoff = 0; warpoff < SIZE; warpoff++) {
      start = rowp[i];
      if (i == (nrows - 1)) { end = nelmts;
      } else { end = rowp[i + 1]; }
      somme = 0;
      // partial reduction
      for (j = start + warpoff; j < end; j += SIZE) {
            somme += (matvals[j] * invec[indx[j]]);
      }
      partial[i * SIZE + warpoff] = somme;
  } // for warpoff
} // for i
```

1D Grid

```
  // finish summing
  for (i = 0; i < nrows; i++) {
    somme = 0.;
    for (k = 0; k < SIZE; k++) {
        somme += partial[i * SIZE + k];
    }
    outvec[i] = somme;
  }
} // spmxv
```

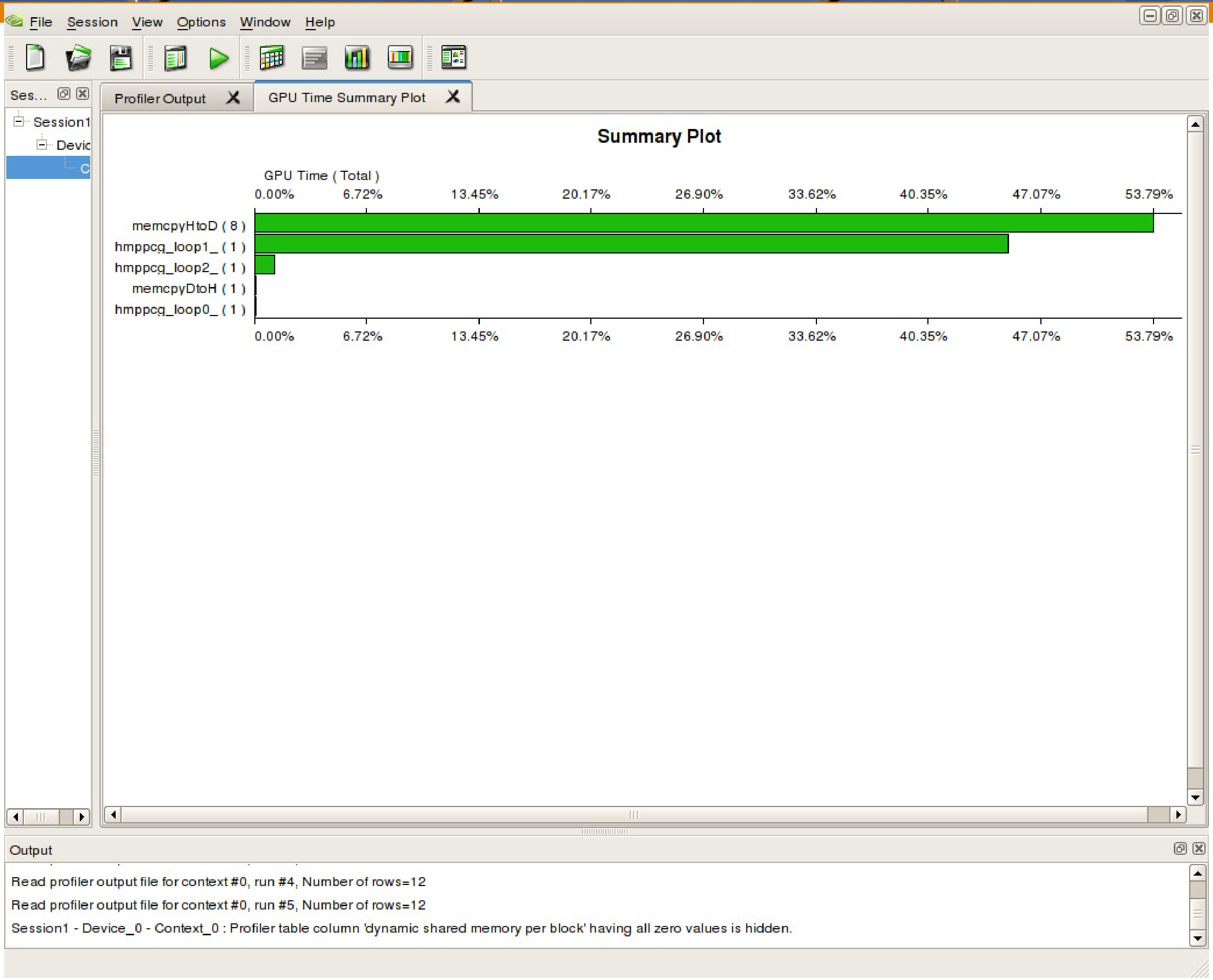
Memory access for mod2as



G. Colin de Verdière, CEA

MOD2AS: results on Uchu

```
Program mod2as: Sparse (CRS) Matrix-vector Multiply
-------------------------------------------------------
 #Rows | #Cols | %Fill |    Time(s)     |    Mflop/s    |OK|
-------------------------------------------------------
    100|     100|  3.50 |          3.1903|           0.000|  T|
    200|     200|  3.75 |        0.086699|           0.035|  T|
    256|     256|  5.00 |        0.086522|           0.076|  T|
    400|     400|  4.38 |        0.086425|           0.162|  T|
    500|     500|  5.00 |        0.086643|           0.289|  T|
    512|     512|  4.00 |        0.087014|           0.241|  T|
    960|     960|  4.50 |         0.08711|           0.953|  T|
   1000|    1000|  5.00 |        0.087103|           1.148|  T|
   1024|    1024|  5.50 |        0.087619|           1.316|  T|
   2000|    2000|  7.50 |        0.098915|           6.066|  T|
   4096|    4096|  3.50 |         0.10218|          11.494|  T|
   4992|    4992|  4.00 |        0.096131|          20.738|  T|
   5000|    5000|  4.00 |         0.09598|          20.838|  T|
   9984|    9984|  4.50 |         0.12392|          72.392|  T|
  10000|   10000|  5.00 |         0.12724|          78.593|  T|
  10240|   10240|  5.72 |         0.13471|          89.077|  T|
```

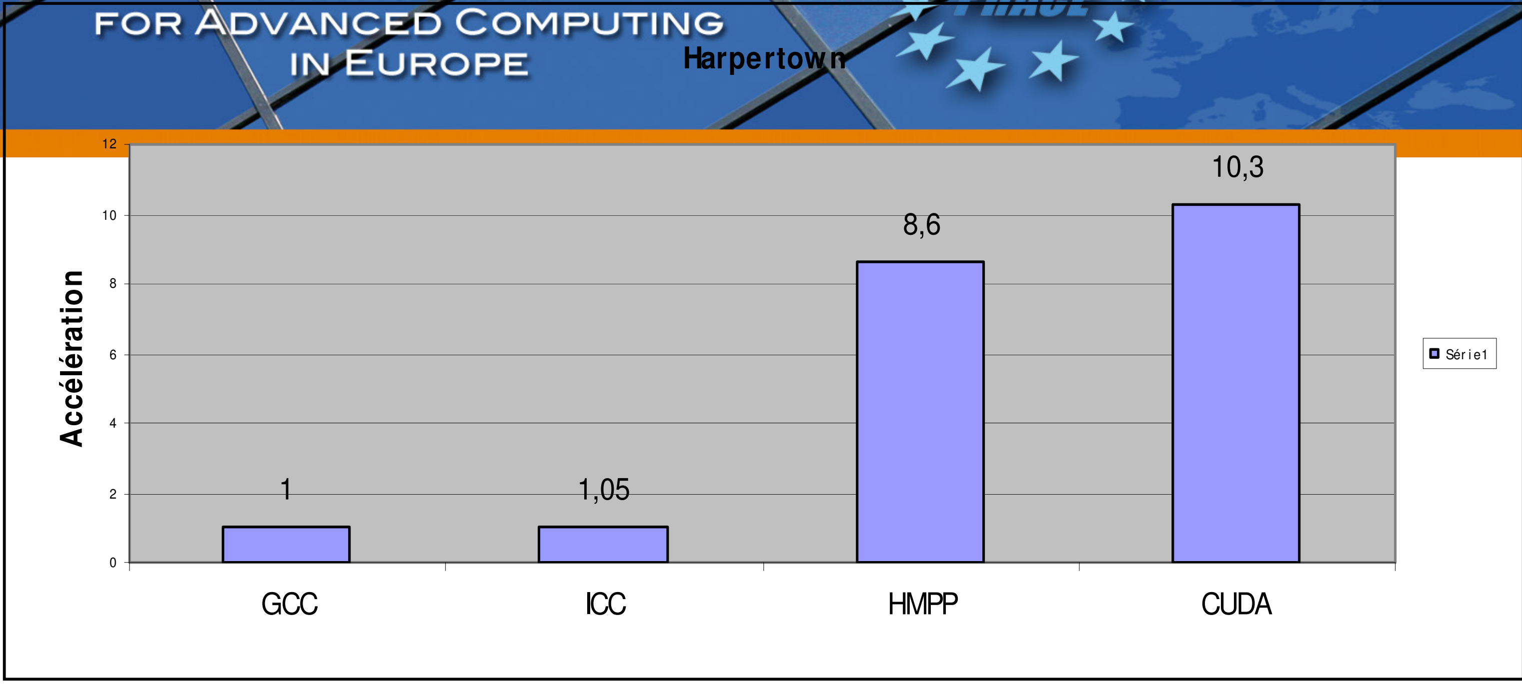


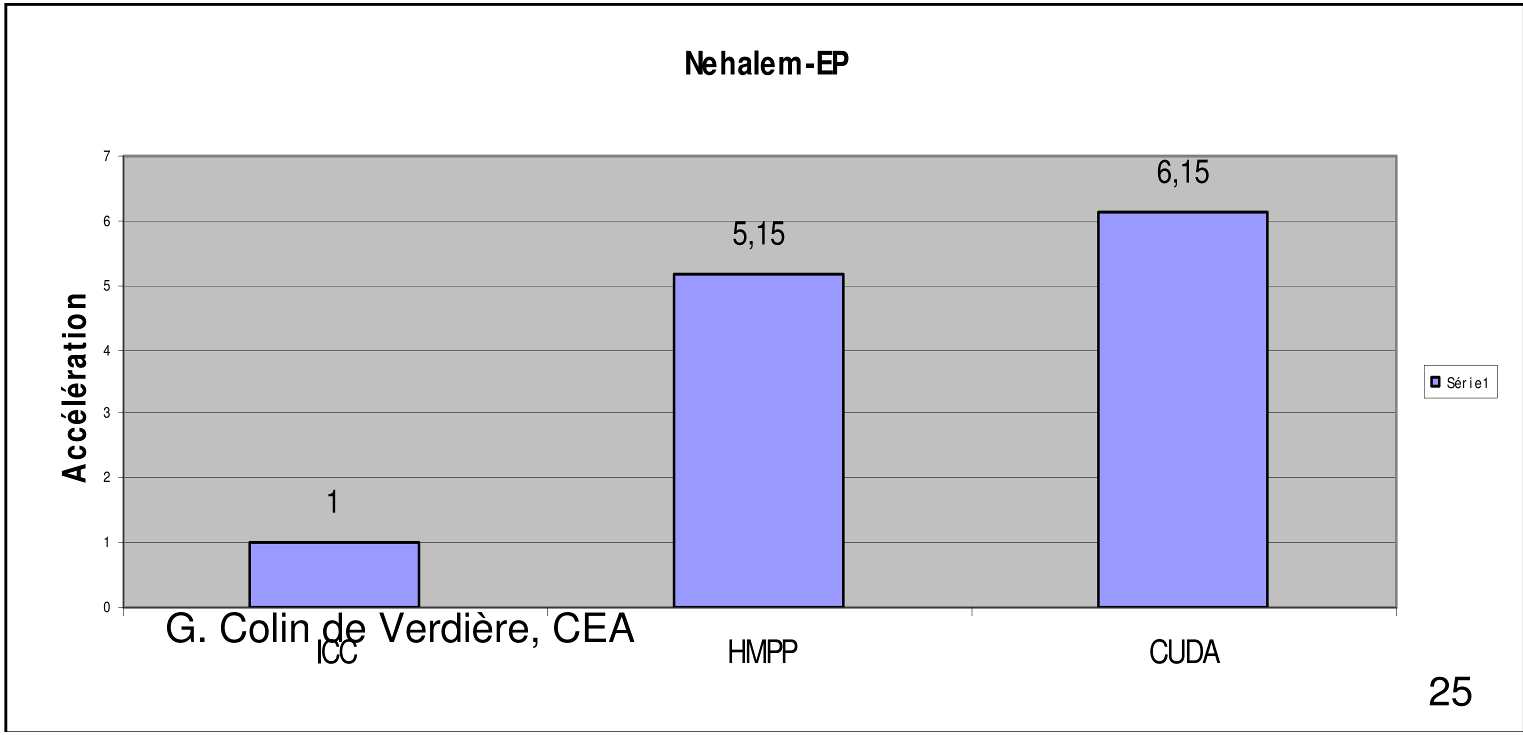
```
Read profiler output file for context #0, run #4, Number of rows=12
Read profiler output file for context #0, run #5, Number of rows=12
Session1 - Device_0 - Context_0 : Profiler table column 'dynamic shared memory per block' having all zero values is hidden.
```

HMPP in a real code

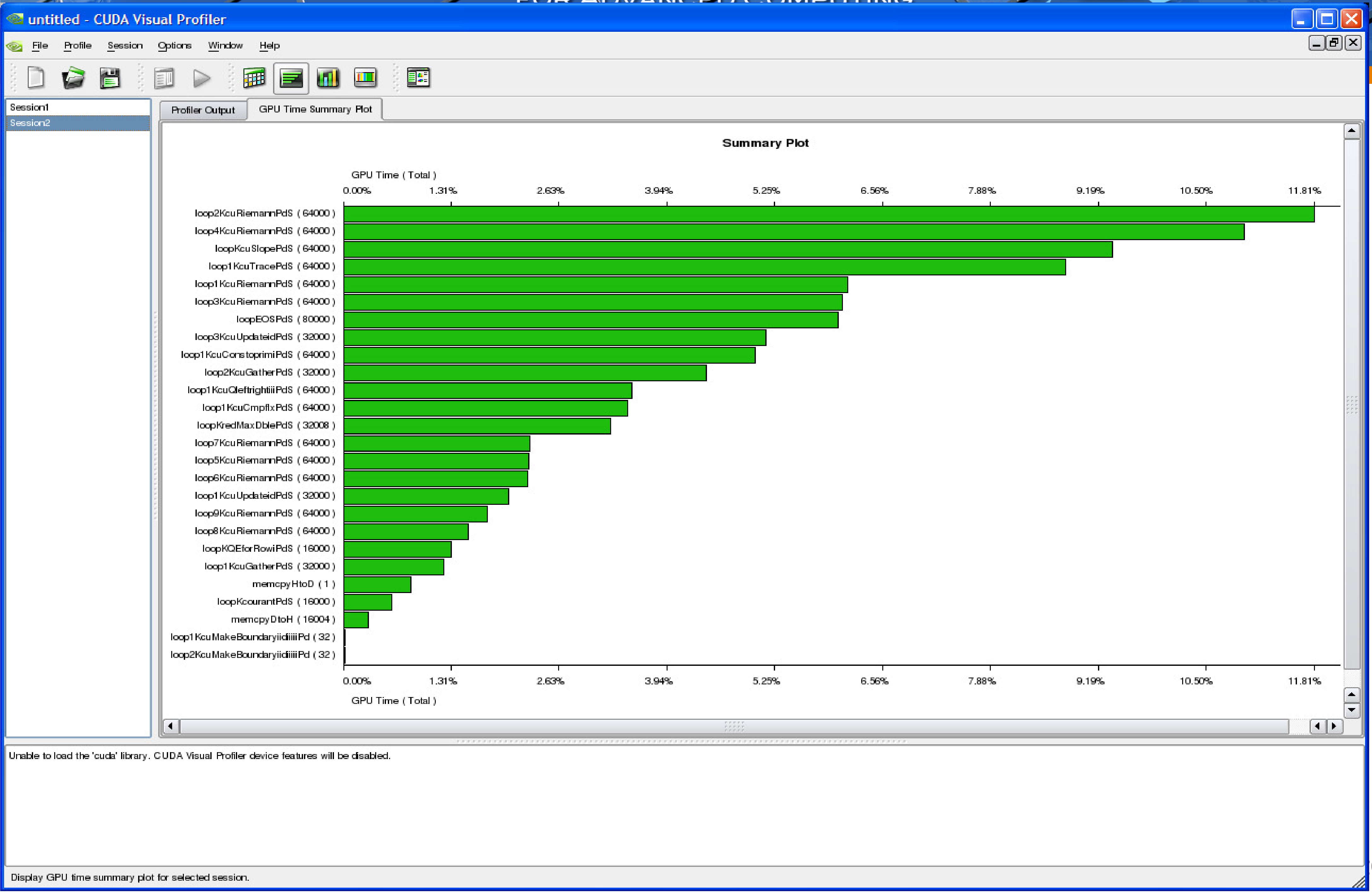
- Test case =10000x10000, 10 iterations
  - DP

Harpertown 2.800Ghz 1626s gcc
Harpertown 2.800Ghz 1540.583s icc
Nehalem EP 2.933GHz 975.590s icc
Tesla T10 1.3GHz 189.087s HMPP
Tesla T10 1.3GHz 158.621s CUDA





G. Colin de Verdière, CEA



Unable to load the 'cuda' library. CUDA Visual Profiler device features will be disabled.

Display GPU time summary plot for selected session.

HMPP port : some conclusions

- Kernels are important to understand a language
  - Yet only full applications can exercise a language
    - Only a small part of HMPP has been tested here

- Port to HMPP is easy
  - The difficulty lies in the data movements

- HMPP allows for decent performances in real cases
  - Asynchronous transfers and calls are key for performance