HSA and GCC

Martin Jambor

SUSE Labs

10th September 2016

Outline

HSA branch: *svn://gcc.gnu.org/svn/gcc/branches/hsa*

Table of contents:

Very Brief Overview of HSA
OpenMP 4.5
Direct HSA
HSA Environment

HSAIL is explicitly parallel (Single Thread Multiple Data)

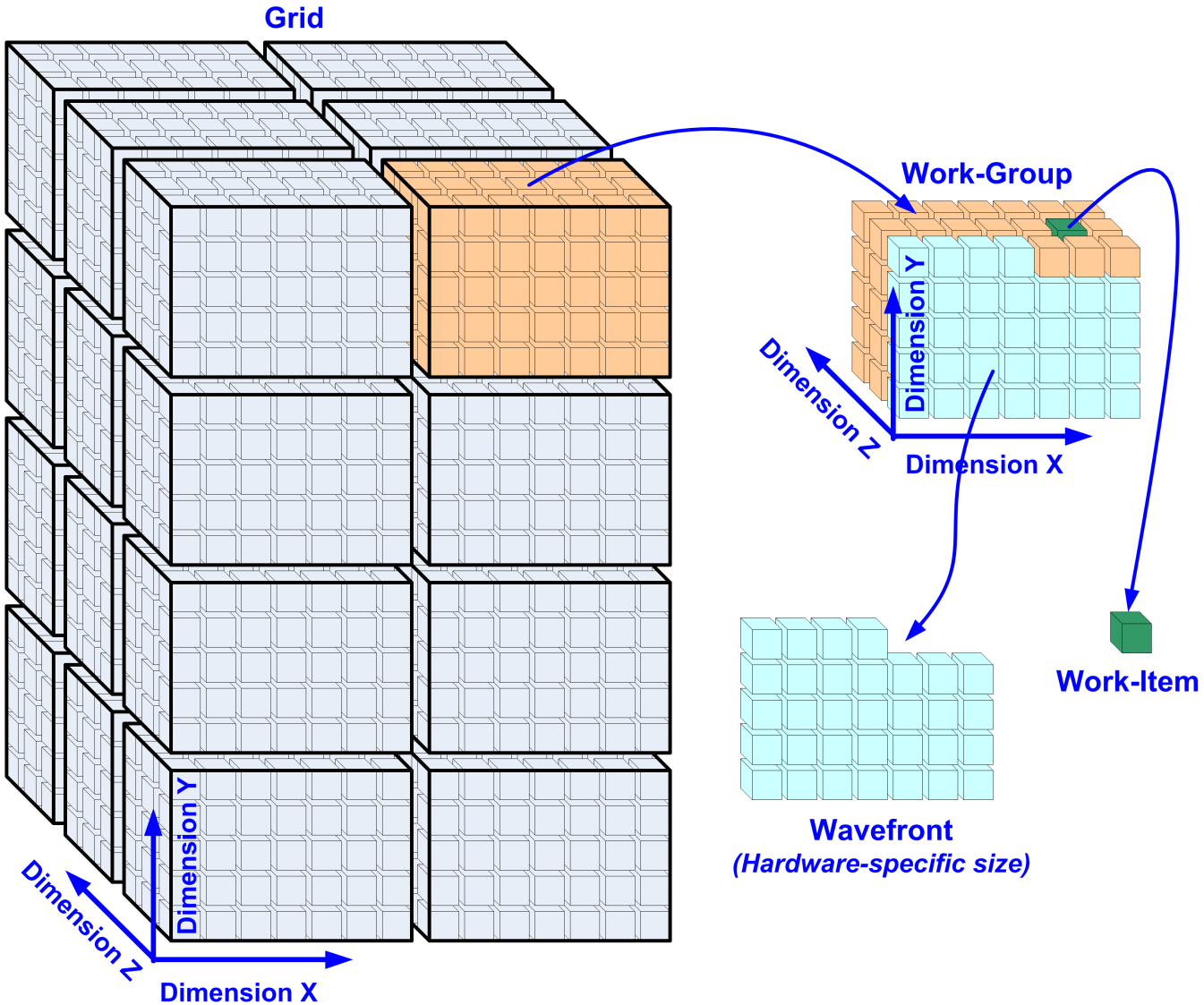


Image from www.hsafoundation.com

Stream

```
#pragma omp target map(to:a[0:N], b[0:N]) map(from:c[0:N])
#pragma omp teams
#pragma omp distribute parallel for private(i)
   for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
```

Stream

```
#pragma omp target map(to:a[0:N], b[0:N]) map(from:c[0:N])
#pragma omp teams
#pragma omp distribute parallel for private(i)
   for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
```

On a CPU:

1. Body of parallel is outlined to a function
2. Body of target up until parallel s outlined to another function
3. Teams and distribute does not really do much on CPUs

Stream

```
#pragma omp target map(to:a[0:N], b[0:N]) map(from:c[0:N])
#pragma omp teams
#pragma omp distribute parallel for private(i)
   for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
```

On a CPU:

1. Body of parallel is outlined to a function
2. Body of target up until parallel s outlined to another function
3. Teams and distribute does not really do much on CPUs

On an HSA GPU:

- We want jut one outlined function (no indirect function calls)

Stream

```
#pragma omp target map(to:a[0:N], b[0:N]) map(from:c[0:N])
#pragma omp teams
#pragma omp distribute parallel for private(i)
   for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
```

On a CPU:

1. Body of parallel is outlined to a function
2. Body of target up until parallel s outlined to another function
3. Teams and distribute does not really do much on CPUs

On an HSA GPU:

- We want jut one outlined function (no indirect function calls)
- The same host-device interface
- Honor all data sharing clauses

Gimplified representation

```
#pragma omp target map(...)
    #pragma omp teams shared(c) shared(b) shared(a)
        #pragma  omp distribute private(i.0)
         for (i.0 = 0; i.0 <= 33554431; i.0 = i.0 + 1)
            #pragma  omp parallel private(i) shared(c) ...
                #pragma  omp for nowait
                 for (i = 0; i <= 33554431; i = i + 1)
                    D.1904 = (long unsigned int) i;
                    D.1905 = D.1904 * 4;
                    D.1906 = a + D.1905;
                    D.1907 = *D.1906;
                    D.1904 = (long unsigned int) i;
                    D.1905 = D.1904 * 4;
                    D.1908 = b + D.1905;
                     ...rest of loop body...
```

Body copy pre-omp-lowering

```
#pragma omp target _griddim_(0:33554432,0) map(...)
    #pragma omp teams shared(c) shared(b) shared(a)
        #pragma  omp distribute private(i.0)
         for (i.0 = 0; i.0 <= 33554431; i.0 = i.0 + 1)
            #pragma  omp parallel private(i) shared(c) ...
                #pragma  omp for nowait
                 for (i = 0; i <= 33554431; i = i + 1)
                    D.1904 = (long unsigned int) i;
                     ...rest of loop body...
    #pragma omp gridified body
      #pragma omp teams phony shared(c) shared(b) shared(a)
          #pragma  omp distribute phony private(i.0)
           for (i.0 = 0; i.0 <= 33554431; i.0 = i.0 + 1)
              #pragma  omp parallel phony private(i) ...
                  #pragma  omp for grid_loop nowait
                   for (i = 0; i <= 33554431; i = i + 1)
                      D.1904 = (long unsigned int) i;
                       ...rest of loop body...
```

Post omp-lowering

```
#pragma omp target _griddim_(0:33554432,0) map(...)
      ...unpack data from host...
     #pragma omp teams shared(c) shared(b) shared(a)
          #pragma  omp distribute private(i.0)
           for (i.0 = 0; i.0 <= 33554431; i.0 = i.0 + 1)
                ...package data for parallel threads...
               #pragma  omp parallel private(i) shared(c) ...
                     ...unpack data...
                    #pragma  omp for nowait
                     for (i = 0; i <= 33554431; i = i + 1)
                         D.1904 = (long unsigned int) i;
                          ...rest of loop body...
     #pragma omp gridified body
           ...unpack data from host...
           #pragma  omp for grid_loop nowait
            for (i = 0; i <= 33554431; i = i + 1)
                D.1904 = (long unsigned int) i;
                 ...rest of loop body...
```

Stream

```
#pragma omp target map(to:a[0:N], b[0:N]) map(from:c[0:N])
#pragma omp teams
#pragma omp distribute parallel for private(i)
   for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
```

Stream

```
#pragma omp target map(to:a[0:N], b[0:N]) map(from:c[0:N])
#pragma omp teams thread_limit(workgroup_size)
#pragma omp distribute parallel for private(i)
   for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
```

Collapse

```
#pragma omp target teams distribute map(...)\
  parallel for collapse(2) \
  shared(a) firstprivate(n) private(i,j)
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      a[i][j] = i + j;
```

Group size of collapsed loops

```
#pragma omp target teams
#pragma omp distribute collapse(2)
  for (int col_start=0; col_start < M; col_start+=BLOCK)
   for (int row_start=0; row_start < N; row_start+=BLOCK)
     {
#pragma omp parallel for collapse(2)
       for (int i=0; i < BLOCK; i++)
         for (int j=0; j < BLOCK; j++)
           {
             int col = col_start + i;
             int row = row_start + j;
             a[col * N + row] = row + col;
           }
     }
```

Matrix multiplication (part one)

```
#pragma omp target teams map(to:A[M*K],B[K*N]) \
 map(from:C[M*N])
#pragma omp distribute collapse(2)
  for (int C_row_start = 0;
       C_row_start < M;
       C_row_start += BLOCK)
    for (int C_col_start = 0;
         C_col_start < N;
         C_col_start += BLOCK)
      {
        /* Each team has a local copy of the following: *
        float As[BLOCK][BLOCK];
        float Bs[BLOCK][BLOCK];
```

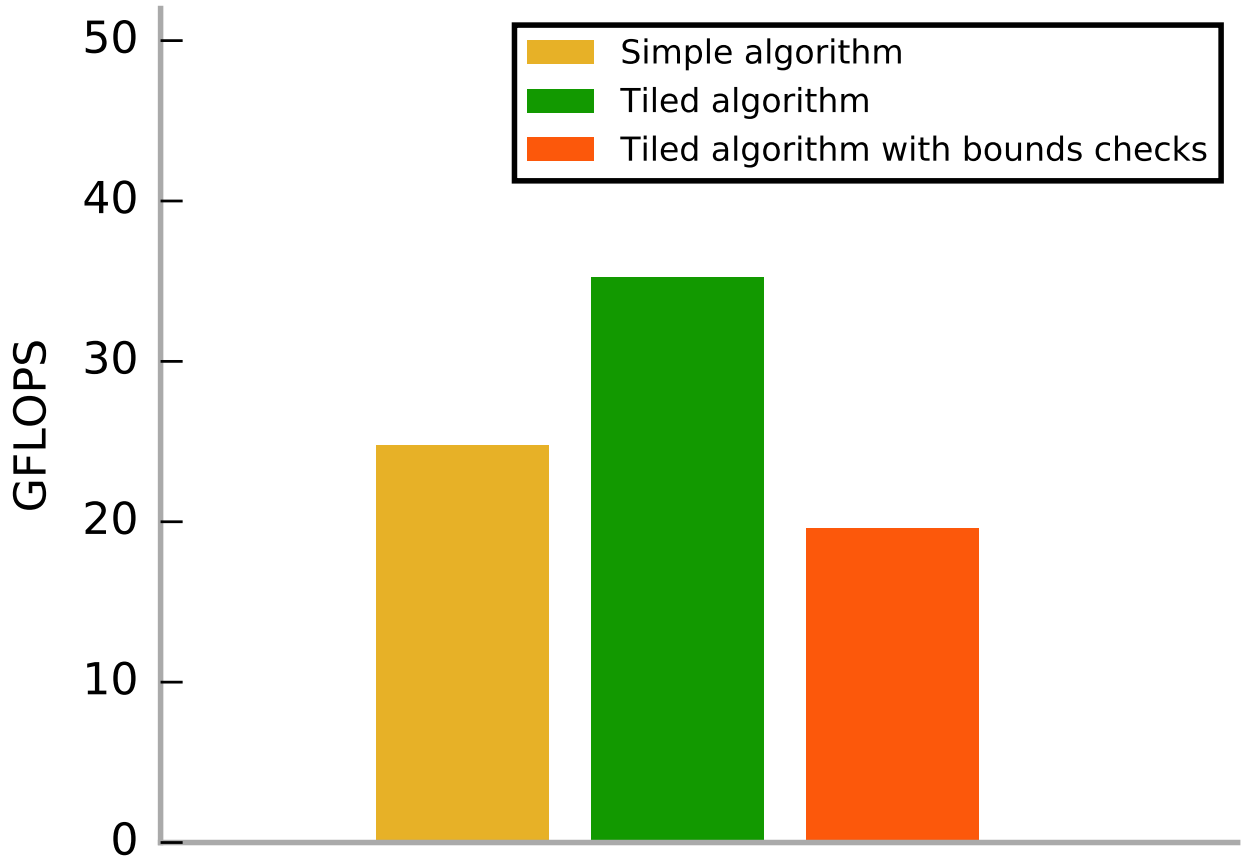
Matrix multiplication (part two)

```
#pragma omp parallel
      {
      int C_row, C_col; float Cval = 0.0;
      for (int kblock = 0; kblock < K; kblock += BLOCK) {
#pragma omp for collapse(2)
          for (int row = 0; row < BLOCK; row++)
            for (int col = 0; col < BLOCK; col++) {
              C_row = C_row_start + row;
              C_col = C_col_start + col;
              As[row][col] = A[(C_row*LDA)+ kblock + col];
              Bs[row][col] = B[((kblock+row)*LDB)+ C_col];
            }
#pragma omp for collapse(2)
          for (int row = 0; row < BLOCK; row++)
            for (int col = 0; col < BLOCK; col++) {
                for (int e = 0; e < BLOCK; ++e)
                  Cval += As[row][e] * Bs[e][col];
            }
      }  /* End for kblock .. */
```

Matrix multiplication (part three)

```
#pragma omp for collapse(2)
        for (int row = 0; row < BLOCK; row++)
          for (int col = 0; col < BLOCK; col++)
            {
              C_row = C_row_start + row;
              C_col = C_col_start + col;
              C[(C_row*LDC)+C_col] = Cval;
            }
        } /* end parallel */
      }  /* end target teams distribute */
```

Performance of tiled matrix multiplication



Please take these numbers with a grain of salt, I did not try to optimize any variant.

Compiler optimization feedback

Use `-fopt-info-openmp[-all]`

Compiler optimization feedback

Use `-fopt-info-openmp[-all]`

- Informs about successful gridifications:

```
stream.c:577:7:  note:  Target construct will be turned into a
gridified GPGPU kernel
```

Compiler optimization feedback

Use `-fopt-info-openmp[-all]`

- Informs about successful gridifications:

```
stream.c:577:7:  note:  Target construct will be turned into a
gridified GPGPU kernel
```

- informs about the reason for failure to gridify:

```
fail.c:287:7:  note:  Will not turn target construct into a
gridified GPGPU kernel because the distribute and an internal loop
do not agree on tile size
fail.c:250:9:  note:  Loop construct does not seem to loop over a
tile size
```

Further OpenMP-related remarks

- I have implemented support for lastprivate
- Need to ignore SIMDs
- Reduction still work in progress
- It would make sense to put gridification into a separate file
- Discrete HSA GPUs will need to have data regions it works on registered
- Gridification for GCN accelerator

Direct HSA (part one)

```
void __attribute__((hsa_kernel))
tiled_kernel (struct kernarg *ka)
{
  static __attribute__((hsa_group_segment))
    float As[BLOCK][BLOCK];
  static __attribute__((hsa_group_segment))
    float Bs[BLOCK][BLOCK];

  int row = __builtin_hsa_workitemid (1);
  int col = __builtin_hsa_workitemid (0);
  int C_row = __builtin_hsa_workitemabsid (1);
  int C_col = __builtin_hsa_workitemabsid (0);
  /* ...unpack stuff from ka parameter... */
  float Cval = 0.0;
```

Direct HSA (part two)

```
  for (int kblock = 0; kblock < K; kblock += BLOCK)
    {
      As[row][col] = A[(C_row*LDA)+ kblock + col];
      Bs[row][col] = B[((kblock+row)*LDB)+ C_col];

      __builtin_hsa_barrier_all ();

      for (int e = 0; e < BLOCK; ++e)
        Cval += As[row][e] * Bs[e][col];

      __builtin_hsa_barrier_all ();
    }
  C[(C_row*LDC)+C_col] = Cval;
}  /* end of kernel function */
```

HSA Environment

Only implementation out there still only by AMD:

- Re-branded as *Radeon Open Compute (ROC)*
- Kernel driver based on 4.4 kernel (1243 patches on top of it)
  - openSUSE Tumbleweed packages:
    http://download.opensuse.org/repositories/home:/jamborm:/roc-1.2/openSUSE_Tumbleweed/
  - Pre-built packages for RedHat and Ubuntu on GitHub
- HSAIL put on the back-burner, AMD promotes GCN
- Run-time to launch GCN kernels is however almost the same

Brig Front-end

- Pekka and his sponsor have copyright assignments almost completed
- I'll do one more round of review. Anybody else?
- Will allow anybody run HSAIL on their CPU
- Eventually should be a basis for a GNU HSAIL finalizer

Brig Front-end

- Pekka and his sponsor have copyright assignments almost completed
- I'll do one more round of review. Anybody else?
- Will allow anybody run HSAIL on their CPU
- Eventually should be a basis for a GNU HSAIL finalizer
- **Can we have it please?**

Thank you.