

GCC support to compile OpenMP 4 target constructs for Heterogeneous System Architecture

Martin Jambor



29th September 2015

Heterogeneous world



Heterogeneous Systems Architecture

The image is a screenshot of a web browser displaying the HSA Foundation website. The browser's address bar shows the URL www.hsafoundation.com. The website's header features the HSA Foundation logo on the left and a navigation menu with the following items: MEMBERSHIP, STANDARDS, DEVELOPERS, HSA ECOSYSTEM, ACADEMIC, and MEDIA INFO. Below the navigation menu is a large green banner with the text "Key Founders of the HSA Foundation". The banner contains a central HSA Foundation logo, which is a shield-shaped emblem with a stylized chip design and the text "HSA FOUNDATION". This central logo is surrounded by two clusters of hexagonal shapes, each containing the logo of a founding company. The left cluster includes ARM, Imagination, TEXAS INSTRUMENTS, and QUALCOMM. The right cluster includes AMD, MEDIATEK, and SAMSUNG. At the bottom of the banner, the text "Key Founders of the HSA Foundation" is written in a white, sans-serif font. Below the banner, a small line of text reads: "Heterogeneous System Architecture (HSA) Foundation is a not-for-profit industry standards body focused on making it dramatically easier to program heterogeneous computing devices. The".

File Edit View History Bookmarks Tools Help

HSA Foundation ARM, AMD,...

www.hsafoundation.com

Harmonizing the Industry around Heterogeneous Computing

HSA FOUNDATION

MEMBERSHIP STANDARDS DEVELOPERS HSA ECOSYSTEM ACADEMIC MEDIA INFO

ARM Imagination

TEXAS INSTRUMENTS

QUALCOMM

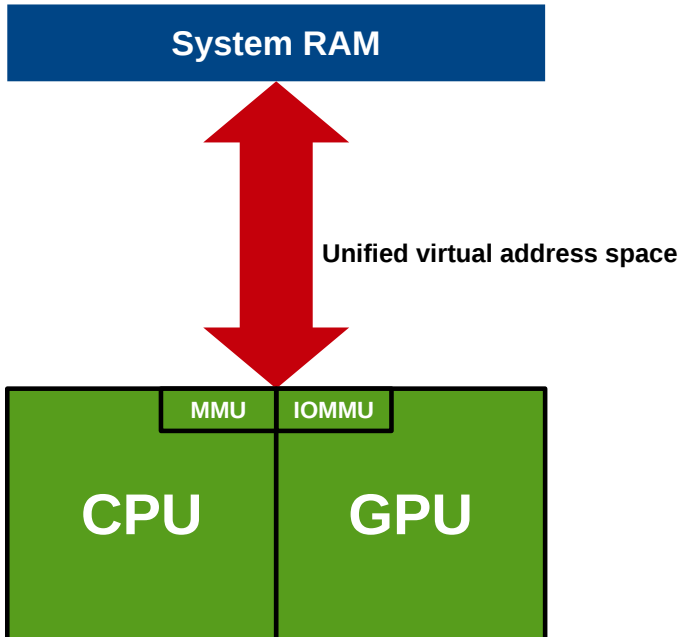
HSA FOUNDATION

AMD MEDIATEK SAMSUNG

Key Founders of the HSA Foundation

Heterogeneous System Architecture (HSA) Foundation is a not-for-profit industry standards body focused on making it dramatically easier to program heterogeneous computing devices. The

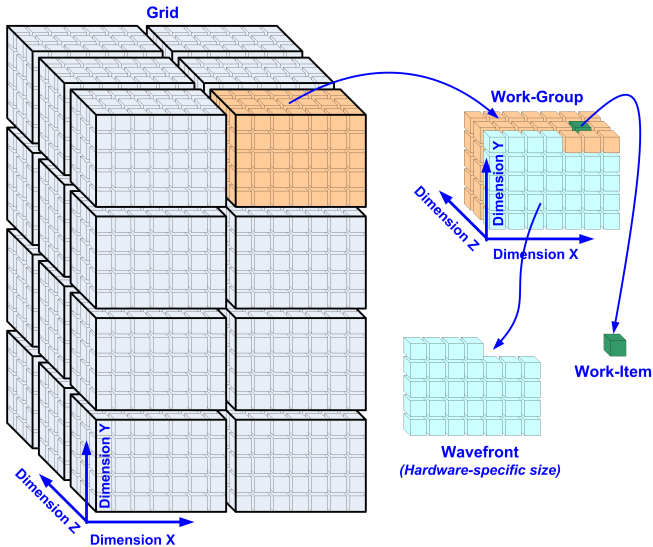
Unified view of memory in HSA



HSA Intermediate Language (HSAIL)

```
prog kernel &__vector_copy_kernel(  
    kernarg_u64 %a,  
    kernarg_u64 %b)  
{  
    workitemabsid_u32 $s0, 0;  
    cvt_s64_s32 $d0, $s0;  
    shl_u64 $d0, $d0, 2;  
    ld_kernarg_align(8)_width(all)_u64 $d1, [%b];  
    add_u64 $d1, $d1, $d0;  
    ld_kernarg_align(8)_width(all)_u64 $d2, [%a];  
    add_u64 $d0, $d2, $d0;  
    ld_global_u32 $s0, [$d0];  
    st_global_u32 $s0, [$d1];  
    ret;  
};
```

HSAIL is explicitly parallel



Getting the compiler and run time

HSA branch:

- ▶ [svn://gcc.gnu.org/svn/gcc/branches/hsa](https://gcc.gnu.org/svn/gcc/branches/hsa)
(also available on the git mirror)

HSA run-time from AMD:

- ▶ <https://github.com/HSAFoundation/HSA-Runtime-AMD>

HSA kernel, firmware, KFDlib from AMD:

- ▶ <https://github.com/HSAFoundation/HSA-Drivers-Linux-AMD>

openSUSE Tumbleweed HSA kernel (at the moment):

- ▶ <https://build.opensuse.org/package/show/home:marxin:carrizo/kernel-default>

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies
1. `./contrib/download_prerequisites`

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies
1. `./contrib/download_prerequisites`
2. `cd ../build`

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies
1. `./contrib/download_prerequisites`
2. `cd ../build`
3. `../src/configure...--enable-offload-targets=hsa
--with-hsa-runtime=/path/to/runtime...`

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies
1. `./contrib/download_prerequisites`
2. `cd ../build`
3. `../src/configure...--enable-offload-targets=hsa
--with-hsa-runtime=/path/to/runtime...`
4. `make && make install`

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies
1. `./contrib/download_prerequisites`
2. `cd ../build`
3. `../src/configure...--enable-offload-targets=hsa
--with-hsa-runtime=/path/to/runtime...`
4. `make && make install`

Compiling your compiler

Nothing to be afraid of:

0. All of <https://gcc.gnu.org/install> still applies
 1. `./contrib/download_prerequisites`
 2. `cd ../build`
 3. `../src/configure...--enable-offload-targets=hsa
--with-hsa-runtime=/path/to/runtime...`
 4. `make && make install`
-
- ▶ Compile with `-fopenmp`
 - ▶ set `LD_LIBRARY_PATH` when running the compiled program
 - ▶ Unlike support for other accelerators, you only need one compiler.

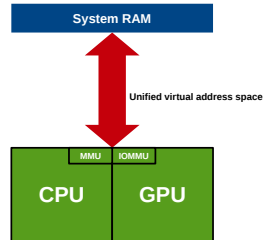
Compiling target constructs

The compiler will attempt to compile each target construct and each function within declare target construct into HSAIL.

Compiling target constructs

The compiler will attempt to compile each target construct and each function within declare target construct into HSAIL.

We have shared virtual address space

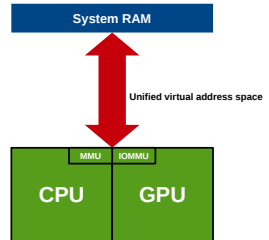


Compiling target constructs

The compiler will attempt to compile each target construct and each function within declare target construct into HSAIL.

We have shared virtual address space

⇒ HSA support can ignore mapping clauses, target data and target update



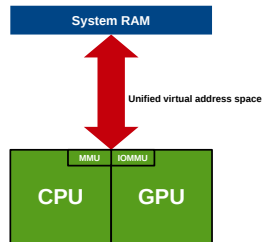
Compiling target constructs

The compiler will attempt to compile each target construct and each function within declare target construct into HSAIL.

We have shared virtual address space

⇒ HSA support can ignore mapping clauses, target data and target update

⇒ Can fall back gracefully on CPU implementations if it could not generate a GPU one



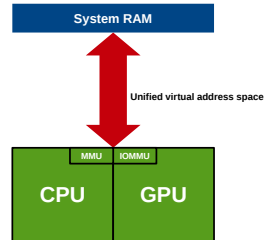
Compiling target constructs

The compiler will attempt to compile each target construct and each function within declare target construct into HSAIL.

We have shared virtual address space

⇒ HSA support can ignore mapping clauses, target data and target update

⇒ Can fall back gracefully on CPU implementations if it could not generate a GPU one



The run-time decides whether (and to which device) offload.

Offloading simple OMP parallel loops

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

The rest of the Stream benchmark loops

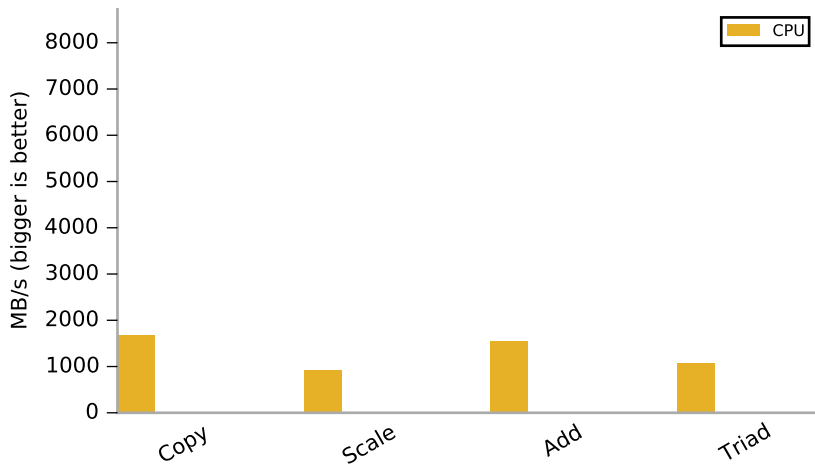
```
/* Scale: */
#pragma omp target
#pragma omp parallel for private(j)
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        b[j] = scalar *c[j];

/* Add: */
#pragma omp target
#pragma omp parallel for private(j)
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        c[j] = a[j]+b[j];

/* Triad: */
#pragma omp target
#pragma omp parallel for private(j)
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        a[j] = b[j]+scalar*c[j];
```

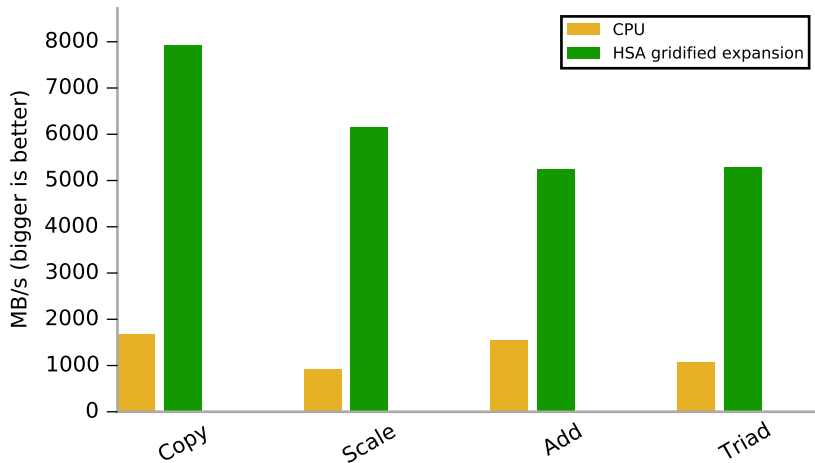
Stream benchmark performance (1)

Stream benchmark results for 64kB arrays (16k of floats) on a Carrizo APU:



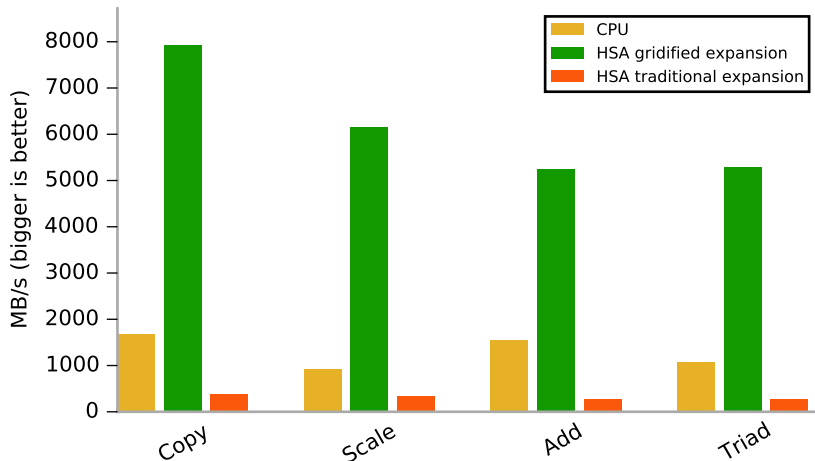
Stream benchmark performance (1)

Stream benchmark results for 64kB arrays (16k of floats) on a Carrizo APU:



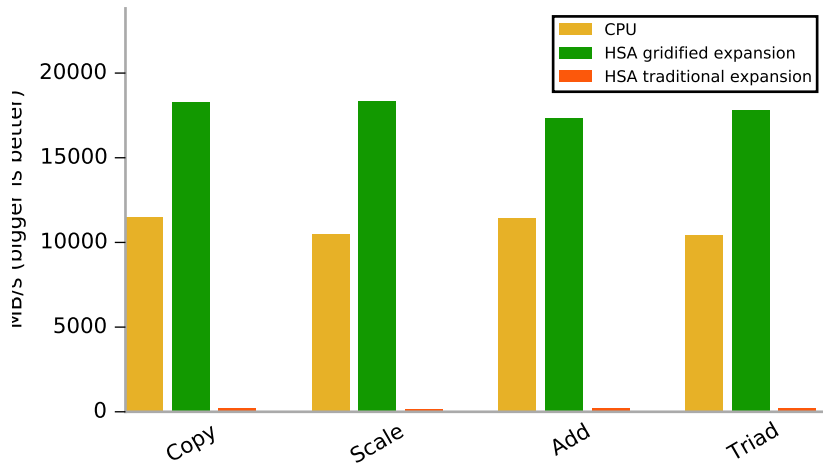
Stream benchmark performance (1)

Stream benchmark results for 64kB arrays (16k of floats) on a Carrizo APU:



Stream benchmark performance (2)

Stream benchmark results for 128MB arrays (32M of floats) on a Carrizo APU:



Gridification

- ▶ Different way of thinking

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

Gridification

- ▶ Different way of thinking

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

- ▶ Still somewhat brittle
 - ▶ Perfect construct nesting required (at IL level, this will have to be relaxed somewhat)
 - ▶ Mechanism of notes to provide feedback to the programmer

Gridification

- ▶ Different way of thinking

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

- ▶ Still somewhat brittle
 - ▶ Perfect construct nesting required (at IL level, this will have to be relaxed somewhat)
 - ▶ Mechanism of notes to provide feedback to the programmer
- ▶ Clauses such as `num_threads` of course prevent gridification

Gridification

- ▶ Different way of thinking

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

- ▶ Still somewhat brittle
 - ▶ Perfect construct nesting required (at IL level, this will have to be relaxed somewhat)
 - ▶ Mechanism of notes to provide feedback to the programmer
- ▶ Clauses such as `num_threads` of course prevent gridification
- ▶ As does non-automatic loop scheduling

Gridification

- ▶ Different way of thinking

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

- ▶ Still somewhat brittle
 - ▶ Perfect construct nesting required (at IL level, this will have to be relaxed somewhat)
 - ▶ Mechanism of notes to provide feedback to the programmer
- ▶ Clauses such as `num_threads` of course prevent gridification
- ▶ As does non-automatic loop scheduling
- ▶ Limited support for teams and distribute constructs

Gridification

- ▶ Different way of thinking

```
/* Copy:*/  
#pragma omp target  
#pragma omp parallel for private(j)  
    for (j=0; j<STREAM_ARRAY_SIZE; j++)  
        c[j] = a[j];
```

- ▶ Still somewhat brittle
 - ▶ Perfect construct nesting required (at IL level, this will have to be relaxed somewhat)
 - ▶ Mechanism of notes to provide feedback to the programmer
- ▶ Clauses such as `num_threads` of course prevent gridification
- ▶ As does non-automatic loop scheduling
- ▶ Limited support for `teams` and `distribute` constructs
- ▶ Reductions through atomics almost done, we plan to support `collapse(2)` and `collapse(3)`

Things we do not intend to support (now)

- ▶ Anything that smacks of a critical section

Things we do not intend to support (now)

- ▶ Anything that smacks of a critical section
- ▶ Sections and tasks and other non-loopy work-sharing

Things we do not intend to support (now)

- ▶ Anything that smacks of a critical section
- ▶ Sections and tasks and other non-loopy work-sharing
- ▶ SIMD constructs (need to change grid size, among other things)

Things we do not intend to support (now)

- ▶ Anything that smacks of a critical section
- ▶ Sections and tasks and other non-loopy work-sharing
- ▶ SIMD constructs (need to change grid size, among other things)
- ▶ Dynamic scheduling

Things we do not intend to support (now)

- ▶ Anything that smacks of a critical section
- ▶ Sections and tasks and other non-loopy work-sharing
- ▶ SIMD constructs (need to change grid size, among other things)
- ▶ Dynamic scheduling
- ▶ Things HSA cannot do (well), e.g. indirect calls

Things we do not intend to support (now)

- ▶ Anything that smacks of a critical section
- ▶ Sections and tasks and other non-loopy work-sharing
- ▶ SIMD constructs (need to change grid size, among other things)
- ▶ Dynamic scheduling
- ▶ Things HSA cannot do (well), e.g. indirect calls

Conclusion

- ▶ HSA and its unified virtual memory
- ▶ About to be programmable with GCC through OpenMP constructs
- ▶ Fallback on CPU implementation a crucial feature
- ▶ Gridification and code duplication

Conclusion

- ▶ HSA and its unified virtual memory
- ▶ About to be programmable with GCC through OpenMP constructs
- ▶ Fallback on CPU implementation a crucial feature
- ▶ Gridification and code duplication

...any questions?