HiHAT Profiling

Samuel Thibault, Asst. Professor, Univ. Bordeaux/INRIA
CJ Newburn, Principal HPC Architect for Compute SW @ NVIDIA

OUTLINE

Part of HiHAT community project

- What can be profiled
- Types of profiling
- Enabling of profiling
- Sample handler
- How it works
- Status summary
- Results
- What’s next

WHAT CAN BE PROFILED

Which activities (actions) to profile

```
typedef enum{
  HHPROF_NONE        =0,          /// Don't profile anything
  HHPROF_TIMESTAMP   =(1<<0),     /// Include time stamps

  HHPROF_ACTIONS     =(1<<1),     /// Any action: kind, state, resource
  HHPROF_COPIES      =(1<<2),     /// Copies: source, dest, size and kind
  HHPROF_ALLOCS      =(1<<3),     /// Allocs: resource, size, MemTraits, chooser
} hhProfOptionEnum;
```

WHAT CAN BE PROFILED

Bit vector values that select which state transitions to profiles

```
typedef enum {
  HH_ON_HOLD          =(1<<0),     /// action is on hold and ineligible to be scheduled
  HH_PENDING_INPUTS   =(1<<1),     /// action is waiting for inputs, & is not schedulable
  HH_SCHEDULABLE      =(1<<2),     /// action is eligible to be schedulable
  HH_IN_EXEC          =(1<<3),     /// action is being executed
  HH_COMPLETED        =(1<<4),     /// action has completed successfully
  HH_HAD_ERROR        =(1<<5),     /// action had an error
  HH_KILLED           =(1<<6),     /// action was killed (not yet supported)
  HH_SUSPENDED        =(1<<7),     /// action was suspended (not yet supported)
   …
} hhActionStateEnum;
```

Red means not yet implemented

TYPES OF PROFILING

Bit vector values that select what happens upon profiling

```
typedef enum{
        HHPROF_OFF          =0,     /// Off
        HHPROF_CALLBACK     =1,     /// Callbacks on
        HHPROF_COUNTER      =2,     /// Count each of the selected ProfOption cases
} hhProfActionEnum;
```

INSTRUCTIONS TO USE PROFILING

- Populate ExecCfg fields

```
typedef struct {  (some fields omitted)
  int  prof_action;   /// One or more profiling actions
  int  prof_option;   /// One or more items for profiling info to record
  int  prof_state;    /// One or more hhActionStates to profile
} hhExecCfg;
```

INSTRUCTIONS TO USE PROFILING

- Register callback function on resources

```
hhRet hhnRegUserFunc(
  void          *func_ptr,       /// function pointer
  hhResrcHndl   resrc_hndl,      /// where this func pointer can execute
  void          *target_config,  /// target-specific config for function
  char          *func_name,      /// NULL-terminated function name
  hhFuncHndl    *out_func_hndl   /// returned function handle
);
```

```
hhnRegUserFunc(
        hhhProfCallbackXXX,
        CPU0_resrc,
        0,
        "hhhProfCallbackXXX",
        &func_hndl_profC);
```

- Register that callback function as a profiling callback function

```
hhRet hhnRegProfCallback(
  int           prof_option,     /// ProfOption(s) this is a handler for
  size_t        num_args         /// 64b args runtime is expected to fill
  hhFuncHndl    func_hndl        /// function handle
  hhExecPol     exec_pol,        /// provide rich set of exec policies
  hhExecCfg     exec_cfg,        /// mode, profiling, exec scope, tag set
  hhResrcHndl exec_resrc         /// target
);
```

```
hhnRegProfCallback(
    HHPROF_ACTIONS|HHPROF_COPIES|HHPROF_ALLOCS,
    7,
    func_hndl_profC,
    dflt_exec_pol,
    dflt_exec_cfg,
    CPU0_resrc);
```

SAMPLE PROFILING CALLBACK HANDLER

```
hhRet hhhProfCallbackXXX(BlobFmt0*blob) {
        size_tnum_args = blob->num_parms;
        void**args = blob->parms;

        int prof_option = reinterpret_cast<int &>(args[0]);
        hhAPIEnum api = reinterpret_cast<hhAPIEnum &>(args[1]);
        int idx = 0;
        printf("ProfOption %d, APIKind %d (%s), state %d (%d is complete), user field %d",
                args[idx++], args[idx++], _hhiAPIName(api),
                args[idx++], HH_COMPLETED, args[idx++]);
        if (prof_option & HHPROF_TIMESTAMP)
                printf(", time %d s,%d us", args[idx++],args[idx++]);
        printf("\n");
```

SAMPLE PROFILING CALLBACK HANDLER, CONT'D

```
    switch (prof_option) {
        case HHPROF_ACTIONS:
          break;
        case HHPROF_COPIES:
          if (idx+3 > num_args)
                printf("Error: Profiling callback registered with too few arguments, %d.\n", num_args);
          printf("  copy from src=0x%x to dst=0x%x, size=0x%x\n", args[idx], args[idx+1], args[idx+2]);
          break;
        case HHPROF_ALLOCS:
          if (idx+2 > num_args)
                printf("Error: Profiling callback registered with too few arguments, %d.\n", num_args);
          printf("  alloc to DataView 0x%x, size %d\n",args[idx], args[idx+1]);
          break;
    }

    return HHRET_SUCCESS;
}
```

SAMPLE: 3 allocs, 1 reg, 3 copies, 1 invoke, 1 clean

```
// Pinned memory on CPU
  hhuAlloc(&CPU_mem_hndl0, &cpu_addr0, alloc_size0, CPU_pinned, def_exec_cfg, CPU0_resrc, NULL, &alloc_hndl0);
// Cleared memory on CPU
  hhuAlloc(&CPU_mem_hndl1, &cpu_addr1, alloc_size1, CPU_clr, def_exec_cfg, CPU0_resrc, NULL, &alloc_hndl1);
// User-allocated memory on CPU
  void* CPU_addr1 = malloc(alloc_size2);
  hhuRegMem(&CPU_mem_hndl2, CPU_addr1, alloc_size2, CPU_mem, def_exec_cfg, CPU0_resrc, NULL, &alloc_hndl2);
// HBM on GPU0
  hhuAlloc(&GPU_mem_hndl0, &gpu_addr0, alloc_size3, GPU_hbm, def_exec_cfg, GPU0_resrc, NULL, &alloc_hndl3);
…
  hhuCopy(CPU_mem_hndl1, CPU_offset1, CPU_mem_hndl0, CPU_offset0, alloc_size1, def_exec_cfg, CPU0_resrc, NULL, &copy_hndl0);
  hhuCopy(CPU_mem_hndl0, CPU_offset1, CPU_mem_hndl2, CPU_offset0, alloc_size1, def_exec_cfg, CPU0_resrc, NULL, &copy_hndl1);
  hhuCopy(GPU_mem_hndl0, GPU_offset, CPU_mem_hndl0, CPU_offset0, alloc_size3, def_exec_cfg, GPU0_resrc, NULL, &copy_hndl2);
// prep invoke
  hhnMkClosure(func_hndlC, blob, 0, &closureC);
// invoke
  hhuInvoke(closureC, def_exec_pol, def_exec_cfg, CPU0_resrc, NULL, &invoke_hndl0);
// clean
  hhuClean(def_tag_set, def_exec_cfg, GPU0_resrc, NULL, &clean_hndl0);
  printf("End of main.\n");
```

RESULTS (1)

```
typedef enum{
  …
  HHPROF_TIMESTAMP = 1,
  HHPROF_ACTIONS   = 2,
  HHPROF_COPIES    = 4,
  HHPROF_ALLOCS    = 8,
} hhProfOptionEnum;
```

ProfOption **3**, APIKind 7 (HHAPI_**HHUALLOC**), state 16 (16 is complete), user field 0, time 311551 s,1505828387 us
ProfOption **9**, APIKind 7 (HHAPI_HHUALLOC), state 16 (16 is complete), user field 0, time 311560 s,1505828387 us
alloc/free/clean of DataView **0x15a36b0**, size 4096
ProfOption 3, APIKind 7 (HHAPI_**HHUALLOC**), state 16 (16 is complete), user field 0, time 311566 s,1505828387 us
ProfOption 9, APIKind 7 (HHAPI_HHUALLOC), state 16 (16 is complete), user field 0, time 311567 s,1505828387 us
alloc/free/clean of DataView **0x15a3720**, size 4096
ProfOption 3, APIKind 10 (HHAPI_**HHUREGMEM**), state 16 (16 is complete), user field -127
7142136, time 311570 s,1505828387 us
ProfOption 3, APIKind 7 (HHAPI_**HHUALLOC**), state 16 (16 is complete), user field 0, time 312186 s,1505828387 us
ProfOption 9, APIKind 7 (HHAPI_HHUALLOC), state 16 (16 is complete), user field 0, time 312188 s,1505828387 us
alloc/free/clean of DataView **0x15a3800**, size 4096
ProfOption 3, APIKind 18 (HHAPI_**HHCCOPYLOCAL**), state 16 (16 is complete), user field 0, time 312192 s,1505828387 us
ProfOption 5, APIKind 18 (HHAPI_HHCCOPYLOCAL), state 16 (16 is complete), user field 0, time 312192 s,1505828387 us
copy from src=0x15a36b0 to dst=0x15a3720, size=4096
ProfOption 3, APIKind 18 (HHAPI_**HHCCOPYLOCAL**), state 16 (16 is complete), user field 0, time 312194 s,1505828387 us
ProfOption 5, APIKind 18 (HHAPI_HHCCOPYLOCAL), state 16 (16 is complete), user field 0, time 312194 s,1505828387 us
copy from src=0x15a3790 to dst=0x15a36b0, size=4096
ProfOption 3, APIKind 19 (HHAPI_**HHCCOPYHETERO**), state 16 (16 is complete), user field 0, time 312214 s,1505828387 us
ProfOption 5, APIKind 19 (HHAPI_HHCCOPYHETERO), state 16 (16 is complete), user field 0, time 312215 s,1505828387 us
copy from src=0x15a36b0 to dst=0x15a3800, size=4096
FROM my_funcCPU: index: 0
FROM my_funcCPU: index: 1
FROM my_funcCPU: index: 2
FROM my_funcCPU: index: 3
FROM my_funcCPU: index: 4
FROM my_funcCPU: index: 5

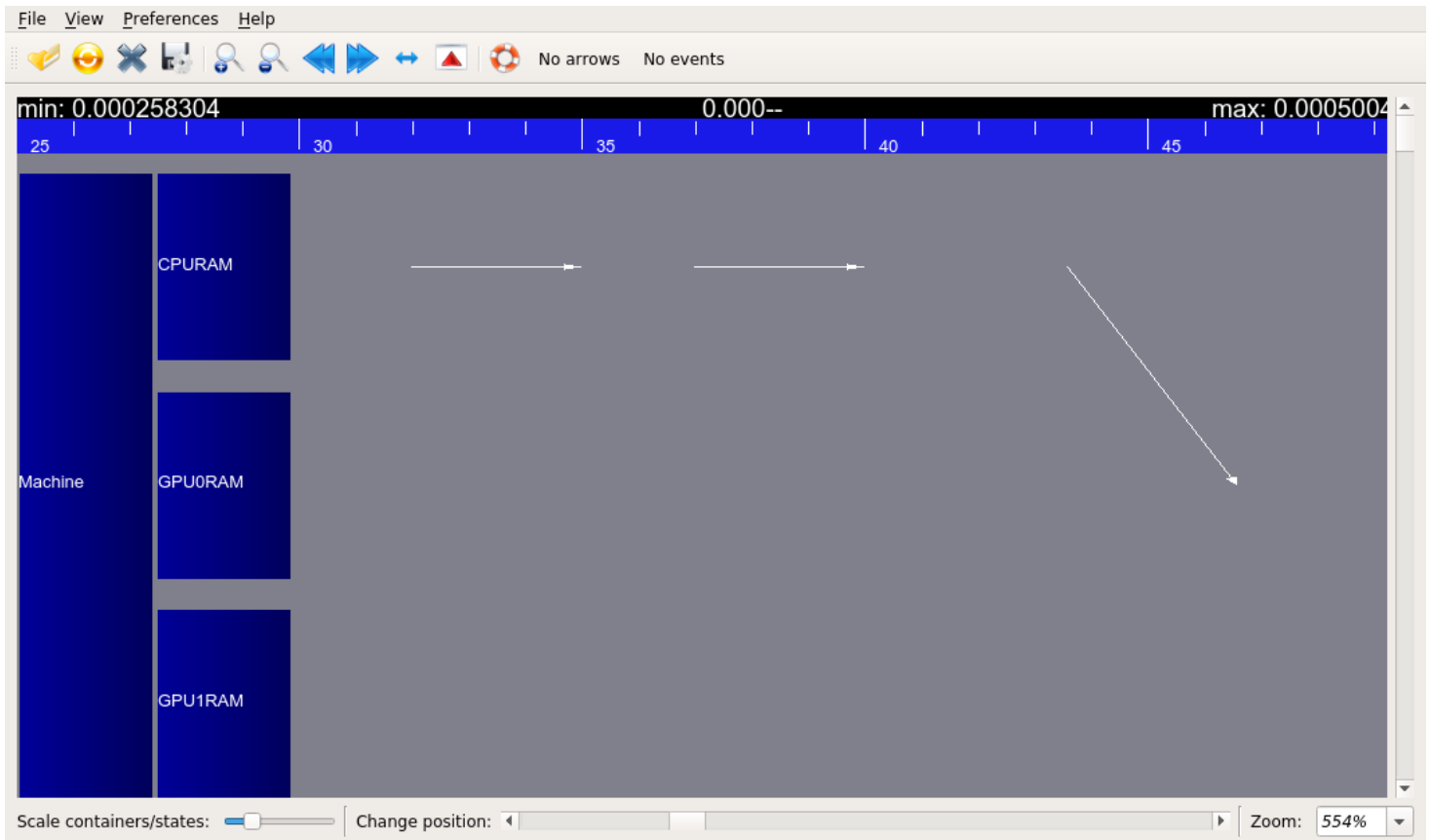
RESULTS (2)

```
typedef enum{
  ...
  HHPROF_TIMESTAMP = 1,
  HHPROF_ACTIONS   = 2,
  HHPROF_COPIES    = 4,
  HHPROF_ALLOCS    = 8,
} hhProfOptionEnum;
```

```
ProfOption 3, APIKind 13 (HHAPI_HHUINVOKE), state 16 (16 is complete), user field 0, time 312217 s,1505828387 us
ProfOption 3, APIKind 8 (HHAPI_HHUCLEAN), state 16 (16 is complete), user field 0, time 312537 s,1505828387 us
ProfOption 9, APIKind 8 (HHAPI_HHUCLEAN), state 16 (16 is complete), user field 0, time 312539 s,1505828387 us
  alloc/free/clean of DataView 0x15a36b0, size 4096
ProfOption 3, APIKind 8 (HHAPI_HHUCLEAN), state 16 (16 is complete), user field 0, time 312540 s,1505828387 us
ProfOption 9, APIKind 8 (HHAPI_HHUCLEAN), state 16 (16 is complete), user field 0, time 312541 s,1505828387 us
  alloc/free/clean of DataView 0x15a3720, size 4096
ProfOption 3, APIKind 8 (HHAPI_HHUCLEAN), state 16 (16 is complete), user field 0, time 312635 s,1505828387 us
ProfOption 9, APIKind 8 (HHAPI_HHUCLEAN), state 16 (16 is complete), user field 0, time 312636 s,1505828387 us
  alloc/free/clean of DataView 0x15a3800, size 4096
End of main.
```

PAJE.TRACE OUTPUT FOR VITE IN STARPU

Standard visualization tools are now able to be built on HiHAT



Courtesy of
Samuel Thibault

POC STATUS SUMMARY

Ready to try; on the path to improvement

- Ready for integration into real profiling tools
- Sample profiler dumps results to stdout
- Supports callbacks; counters partially supported but not tested
- Supports any actions, allocs, copies; reports time stamps
- Supports profiling upon completion of action only
- Doesn’t support independent callbacks of the same kind to multiple tools

WHAT'S NEXT

- Try it out
  - StarPU
  - Allinea
  - You!
- Track increasing support in HiHAT, e.g. multiple action states
- Counters – try and robustify, if you like – it'd be easy