NUMAPROF

A NUMA MEMORY PROFILING TOOL
PLAN

- Reminder & what we want
- NUMAPROF internals
- GUI and example
- Conclusion
Reminder on NUMA
Today topology

- Example current *Intel Knight Landing*, mode *SNC2* or *SNC4*
- Also add fast memory *MCDRAM* presented as *NUMA* or *LLC cache*
Implicit binding: first touch

- New allocated segments are **physically empty**
- They are filled on **first touch**
- Page selection **depend** of the **thread position**
Typical OpenMP mistake

- Make first **init outside of OpenMP** (in thread 1)
- So **each pages** will be first touched **on NUMA 1**

```c
#pragma omp parallel for
for (int i = 0 ; i < SIZE ; i++)
    array[i] = 0;
```

- Then access

```c
#pragma omp parallel for
for (int i = 0 ; i < SIZE ; i++)
    array[i]++;
```

- **Bad performance** due to remote accesses!
Wish list for a profiling tool...

- We want to know if we make remote accesses
- Ideally we need to know where...
- We can dream, we want to know which allocation contain issues
- We want to know where the first touch has been done
- On KNL we want to check MCRAM accesses
Existing tools

- **MemProf** [1], a research paper from Grenoble
  - Hardware feature provided by AMD & kernel module, No GUI
- **SNPERF** [2], again a research project
  - Link utilization on time chart
- **NUMAgrind** [3]
  - Looks nice but not available
- **Numatop**
  - Similar to top, global profiling
- **HPCToolkit** [4]
  - Also have a nice interface, but hw counters & sampling

---

[4] A Tool to Analyze the Performance of Multithreaded Programs on NUMA architectures
NUMPROF

HOW TO KNOW IF WE ARE RIGHT IN A REAL APPLICATION?
NUMAPROF

- Take back the idea from MALT
  - Web interface
  - Source annotation
  - Global metrics

- Use intel Pin
  - Permit to instrument all memory accesses
  - Parallel opposite to valgrind
  - Difficulty: we cannot easily use libs inside the tool
  - I would have used hwloc and libnuma.....
On access we need...

- On each access we want to know if it is
  - Remote access
  - Local access
  - MCDRAM access
  - Page is pinned
  - Thread is pinned

- So, we need to know
  - Where is the page
  - Where is the current thread

- We can skip accesses to local stack (overhead 80x -> 40x)
Keep track of page mapping

- Can query page location with

```c
int status
long ret = move_pages(0,1,pages,\textbf{NULL},&status,0);
```

- It cost a system call

- Cannot do it for every access

- Need to build a cache
Similar to kernel page table

- I use the same layout than kernel page table
  - With multiple levels of 512 entries

- For each page we track
  - NUMA location
  - If has already been touched
  - If first touch was from the binded or not binded thread

- Need to track `mmap/mremap/munmap`
  - To update page touched status
Shadow Page table

9 bits = 512 entries

- NUMA page
- Touched
- bind
Allocation site

- Global Directory
- Upper Directory
- Middle Directory
- Page Table Entry

Address

9 bits = 512 entries

Malloc call-site counters

Offset
Limit mutexes & atomics

- I use caches to **accumulate locally** and **flush** sometimes

![Diagram](attachment:diagram.png)

- Thread state
  - Local Instruction Cache
    - 0xA8E5
    - Counters

- Process state
  - Global Instruction Map
    - 0xA8E5
    - Counters
    - Mutex
Overhead and scalability

- Of course overhead is large: ~30x
- But is scale
- Example code hydro on KNL:
GUI and example
Global summary

First touch

Memory access

Access matrix

Peak allocated numa pages
Statistics per thread
OMP and huge pages

- Huge pages & thread splitting
- Most of the time do not match exactly
- Not a big issue if limited
Details per thread

**Thread 4**
- **Lifeline**: 64.31% → 90.05%
- **CPU thread binding**: 4
- **Numa thread binding**: 0
- **Numa mem. policy**: MPOL_DEFAULT on -1 considered as NO_BIND

**Accesses**

**Placing log**
- At 64.31%, pin thread on node 0
- At 64.31%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND

---

**Thread 5**
- **Lifeline**: 64.32% → 90.91%
- **CPU thread binding**: 5
- **Numa thread binding**: 1
- **Numa mem. policy**: MPOL_DEFAULT on -1 considered as NO_BIND

**Accesses**

**Placing log**
- At 64.32%, pin thread on node 1
- At 64.32%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND
### Source & asm annotations

**Line 41**

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pinned first touch</td>
<td>50,290</td>
</tr>
<tr>
<td>Unpinned first touch</td>
<td>910</td>
</tr>
<tr>
<td>Local</td>
<td>49,269,280</td>
</tr>
<tr>
<td>Remote</td>
<td>1,855,585</td>
</tr>
<tr>
<td>Unpinned page</td>
<td>0</td>
</tr>
<tr>
<td>Unpinned thread</td>
<td>377,860</td>
</tr>
<tr>
<td>Unpinned both</td>
<td>932,461</td>
</tr>
<tr>
<td>MCDRAM</td>
<td>0</td>
</tr>
<tr>
<td>Non allocated</td>
<td>0</td>
</tr>
</tbody>
</table>
Code Hydro

- KNL Without HBM
- WITH HBM

<table>
<thead>
<tr>
<th>Memory access</th>
<th>Local</th>
<th>Remote</th>
<th>Unpinned page</th>
<th>Unpinned thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNL Without HBM</td>
<td></td>
<td></td>
<td>26%</td>
<td>74%</td>
</tr>
<tr>
<td>WITH HBM</td>
<td></td>
<td></td>
<td>15%</td>
<td>41%</td>
</tr>
</tbody>
</table>

Sebastian Vidal
15/02/2018
Original Hydro access matrix
#pragma omp parallel for private(i) if (m_numa) SCHEDULE
for (int32_t i = 0; i < m_nbtiles; i++) {
    int t = m_mortonIdx[i];
    m_tiles[t] = new Tile;
}
Non parallel allocations

```cpp
ThreadBuffers::ThreadBuffers(int32_t xmin, int32_t xmax, int32_t t
{  
  int32_t lgx, lgy, lgmax;
  lgx = (xmax - xmin);
  lgy = (ymax - ymin);
  lgmax = lgx;
  if (lgmax < lgy)
    lgmax = lgy;

  m_q = new Soa(NB_VAR, lgx, lgy);
  m_qxm = new Soa(NB_VAR, lgx, lgy);
  m_qxp = new Soa(NB_VAR, lgx, lgy);
  m_dq = new Soa(NB_VAR, lgx, lgy);
  m_qleft = new Soa(NB_VAR, lgx, lgy);
  m_qright = new Soa(NB_VAR, lgx, lgy);
  m_qgdnv = new Soa(NB_VAR, lgx, lgy);

  m_c = new Matrix2<real_t> (lgx, lgy);
  m_e = new Matrix2<real_t> (lgx, lgy);```

Parallel allocations

- Original

```cpp
for (int32_t i = 0; i < m_numThreads; i++) {
    m_buffers[i] = new ThreadBuffers(...);
    assert(m_buffers[i] != 0);
}
```

- Modified

```cpp
#pragma omp parallel
{
    int i = omp_get_thread_num();
    #pragma omp critical
    m_buffers[i] = new ThreadBuffers(...);
    assert(m_buffers[i] != 0);
}
```
Speed up obtained on Hydro

Access matrix

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Unpaired</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time (s)

- WITHOUT HBM
- Before: 45 - 29 = 16
- After: 45 - 18 = 27

-18%
Remaining consts

<table>
<thead>
<tr>
<th>Search</th>
<th>5.1G do_cpln</th>
<th>1.4G</th>
<th>160.6M</th>
<th>83.7M</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Line 1195</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pinned first touch</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpinned first touch</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local</td>
<td>72 876 032</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote</td>
<td>219 123 712</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpinned page</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpinned thread</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpinned both</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCDRAM</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non allocated</td>
<td>291 999 744</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Tool easy to use coming with a nice GUI
- Useful to check an application.

- Of course, still a lot of works to do
  - Add support of call stacks
  - Consider cache simulation
  - Optimizations
  - Time charts

- Also want to support DynamorRIO and valgrind.

http://memtt.github.io/
How to control memory placement

**FIRST SOLUTION:** BIND THE PROCESS AT LAUNCH TIME

- **BIND CPU**
  
  ```
  hwloc-bind --cpubind node:0 ./mycommand
  ```

- **BIND MEMORY**
  
  ```
  hwloc-bind --membind node:0 ./mycommand
  ```

- **NUMACTL OPTIONS**
  
  ```
  numactl --membind -m 0 ./mycommand
  ```

But what if we have 4 NUMA NODES? We need to control at thread level.
Thread binding

```c
#include < sched.h >

cpu_set_t cpuset;
CPU_ZERO(&cpuset);
CPU_SET(1,&cpuset);

pthread_set th = pthread_self();
pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
```

**KMP_AFFINITY=scatter ./mycommand**

**OMP_PROC_BIND=TRUE ./mycommand**
Extracted metrics

- First touch
  - Pinned first touch
  - Unpinned first touch

- Accesses
  - Local access
  - Remote access
  - Unpinned thread access
  - Unpinned page access
  - Unpinned both access
  - MCDRAM access
Performance of this example

- Memset: 35
- OMP Loop: 15

This is true only if we bind the threads.

Example of run on 8 threads, 2 NUMA nodes
What is NUMA?

- Each CPU has its own memory
- Access to remote memory we need to go through the owner CPU
Want to link to allocation site

- I want to provide statistics on allocation site
- Need on each access to know where the bloc was allocated
- Add entries into the page table
- Split page into blocs of 8 bytes
- Store a pointer for each bloc to point the segment descriptor
- Issue if allocation are smaller then 8 bytes (Incompatible with jemalloc)