Report: hierarchical workstealing in XKAAPI

XKAAPI team

Contents

1 Introduction

TODO: describe the need for HWS

2 Enabling HWS in XKAAPI

2.1 Build configuration

No new configuration has been added to the build system. However, the runtime has to be compiled with *hwloc* and *numa* support for HWS to be enabled:

```
./configure --with-hwloc --with-numa
```

2.2 Environment variables

The steal request emission routine has to be specialized by setting the *KAAPI_EMITSTEAL* environment variable to "hws":

```
$> KAAPI_HWS_LEVEL=hws ./a.out
```

Specific memory heriarchy levels can be used by setting the *KAAPI_HWS_LEVELS* environment variable. It consists of a comma separated list of one or more of the following values:

- ALL: enables all the levels,
- NONE: disables all the levels,
- L3: enable the L3 cache level,
- NUMA: enable the numa level,
- SOCKET: enable the socket level,
- MACHINE: enable the machine level,
- FLAT: enable the flat level.

For instance:

```
$> KAAPI_HWS_LEVEL=hws KAAPI_HWS_LEVELS=FLAT,NUMA ./a.out
```

Not setting this variable enables the NUMA, SOCKET, MACHINE and FLAT memory levels.

3 HWS implementation in XKAAPI

3.1 Overview

The *src/hws* directory has been added to the runtime sourcecode. It implements:

- subsystem initialization (*kaapi_hws_initialize.c*),
- task pushing (*kaapi_hws_pushtask.c*),
- steal request emission (*kaapi_hws_emitsteal.c*),
- adaptive task handling (*kaapi_hws_adaptive.c*),
- initial splitter (*kaapi_hws_splitter.c*),
- workstealing queue implementation (*kaapi_ws_queue_xxx.c*),
- related performance counters (*kaapi_hws_counters.c*),
- scheduler synchronization (*kaapi_hws_sched_sync.c*).

3.2 Hierarchy construction

TODO

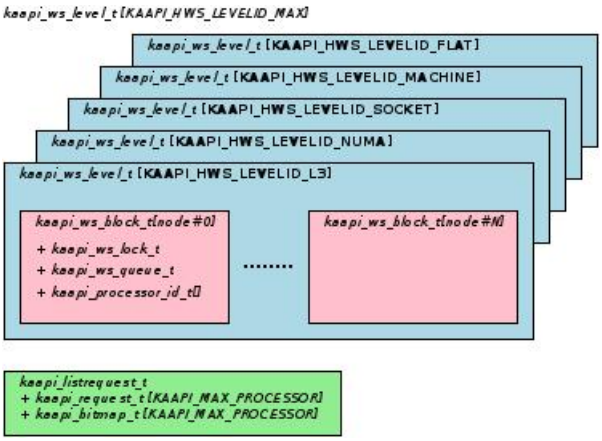

Figure 1: HWS data structure relations

3.3 Steal request emission algorithm

The steal request emission entrypoint is the kaapi_hws_emitsteal routine. It is still in progress, the final implementation will depend on the performances achieved during the benchmarking process. Currently, the algorithm is as follow:

1. try to pop from the local queue, stop upon success.
2. steal in each parent level.
3. during the parent level iteration, try to steal in each child level.
4. if child level stealing fails, try to steal in parent level leaves.
5. fail the unanswered requests.

3.4 Workstealing queue virtualization

To provide better flexibility regarding workstealing decision, a new *queue* interface has been added. It contains the following methods:

- push: push a task in the queue,
- pop: pop a task from the local queue,
- steal: reply to a set of steal requests.

Currently, there is only a basic static lifo implementation.

3.5 Emisteal routine virtualization

The *emitsteal* routine has been virtualized. The *KAAPI_EMITSTEAL* environment variable controls the implementation used:

- hws: use the hierarhical workstealing implemented by the *kaapi_hws_sched_emitsteal* routine,
- any other value: use the default *kaapi_sched_emitsteal* routine.

3.6 Workstealing blocks

TODO

4 Benchmarks

TODO