NUMA-aware CPU core allocation in cooperating dynamic applications

Jiri Dokulil, Siegfried Benkner

Faculty of Computer Science

University of Vienna, Vienna, Austria

universität wien

FWF Der Wissenschaftsfonds.

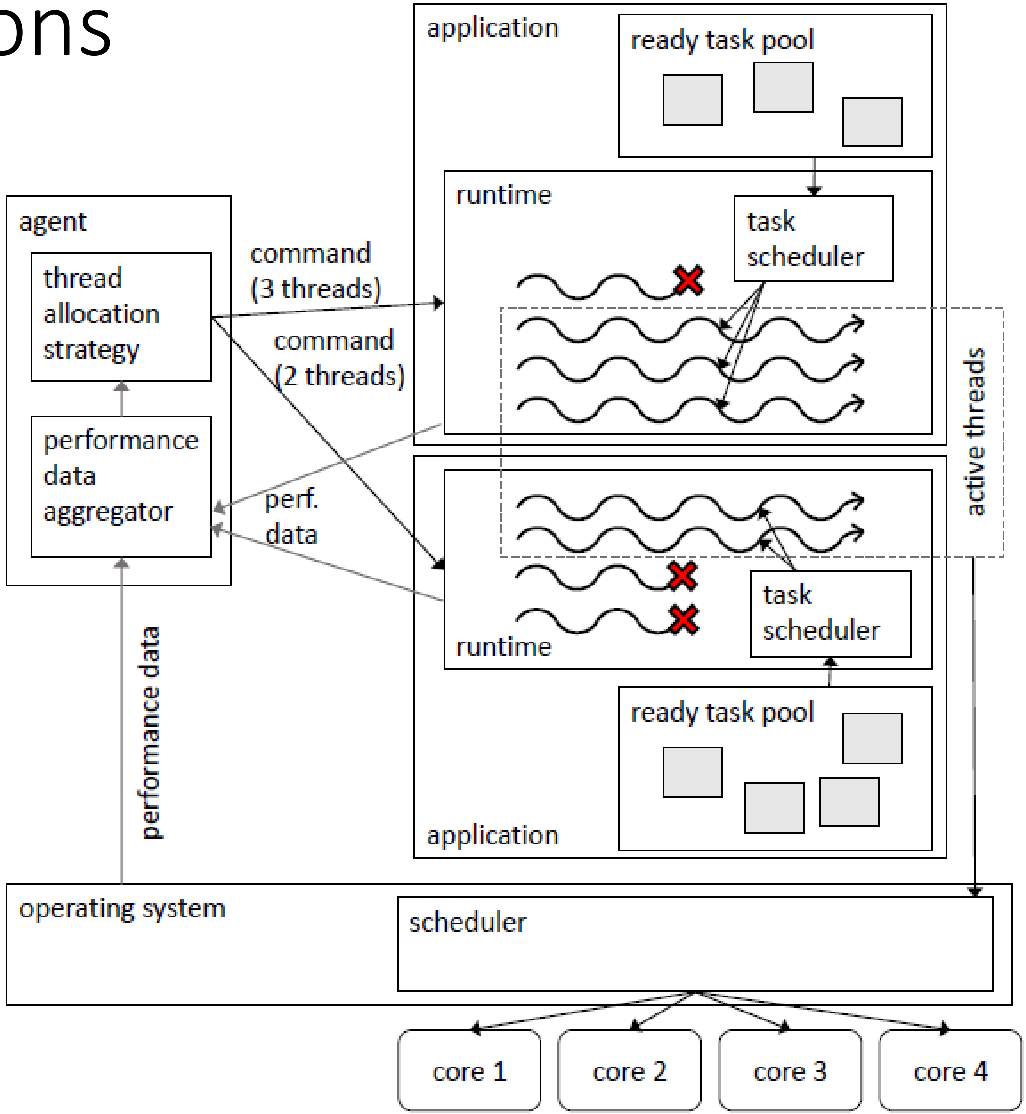
Cooperating applications

- multiple applications running concurrently working together (by sharing data and coordinating their progress) towards a common goal
- producer-consumer; in-situ simulation and visualization; multi-physics codes, …
- dynamic
  - applications built using a task-based runtime system
  - can quickly adapt to performance variability and changing resource availability

CPU core allocation

- for cooperating applications resource arbitration is necessary to decide what cores are available to the different applications
- normally handled by the operating system
- can be dynamic and change over time
- Can we improve it with the extra information that the runtime system has?
  - OCR-Vx in our case (Open Community Runtime)
  - any task-based runtime system in general
  - worker threads used to run tasks
    - scheduled by the runtime system
- potential space for improving on what the OS can do:
  - eliminate oversubscription
  - limit resources of poorly scaling applications
  - ensure even progress of connected applications
    - producer/consumer
  - quickly move resources to another application performing a subtask
    - fine-grained components spread across different processes

Architecture of cooperating OCR-Vx applications



NUMA

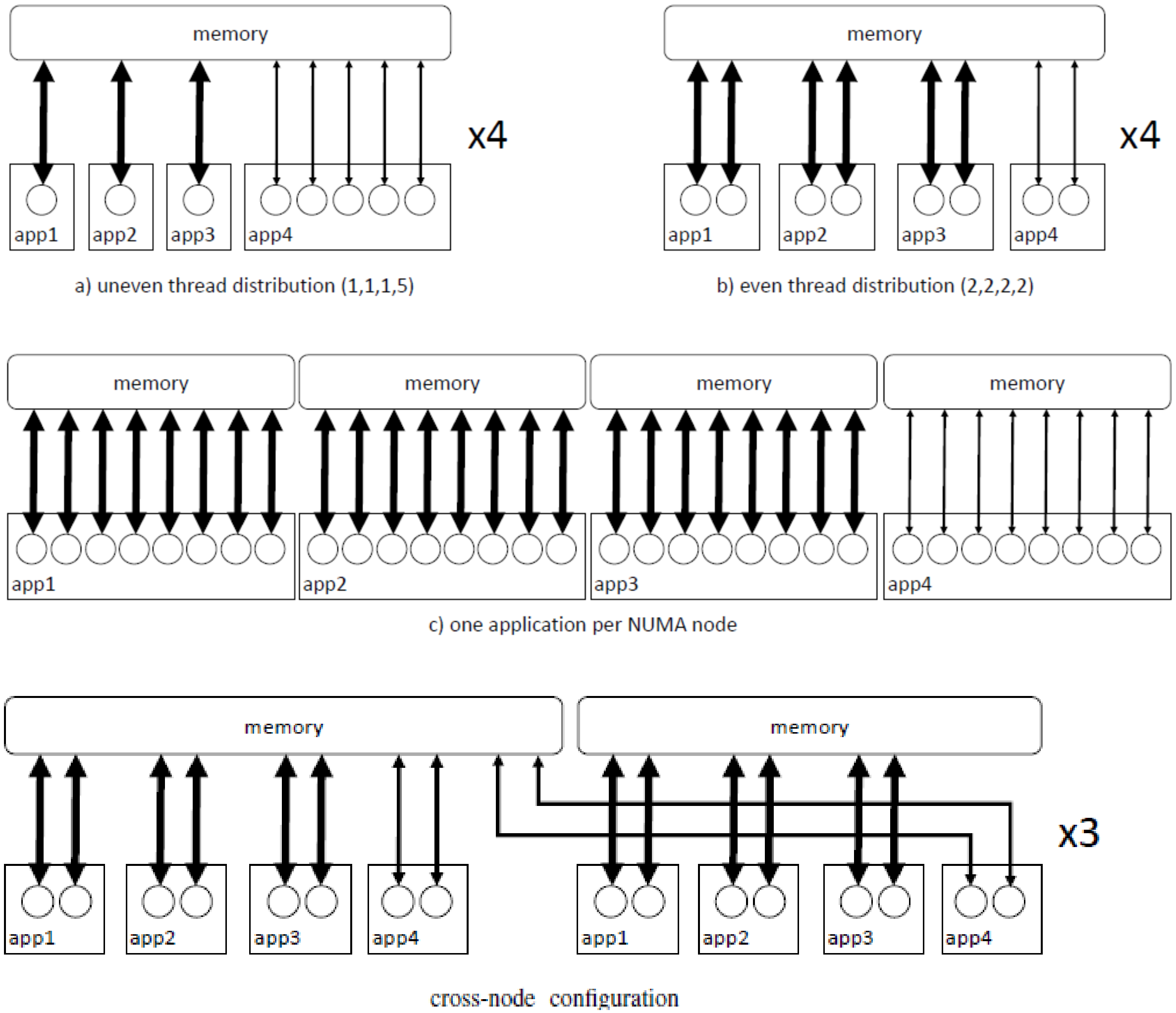
- multiple applications on one machine → NUMA is important
- different ways of interpreting “use 8 cores”
  1. total number across all NUMA nodes
  2. activate/deactivate individual cores (bitmap)
  3. number of threads per NUMA node
     - 2/2/2/2
     - 8/0/0/0
     - 4/0/4/0
  - implemented in OCR-Vx
    - possible in most other runtime systems

Model

- based on roofline model
- NUMA-aware
- fixed thread allocation
  - we model momentary state, not the whole execution
- multiple applications competing for memory access
  - model based on observations (STREAM benchmark)
  - basically, each core gets at least its fair share, the rest is split among those that need more

	memory-bound	compute-bound
arithmetic intensity (AI)	0.5	10
number of instances	3	1
threads per NUMA node	1	5
peak memory bandwidth per thread (peak GFLOPS / AI)	$10/0.5 = 20$	$10/10 = 1$
peak memory bandwidth per instance (per-thread * #threads)	$20 * 1 = 20$	$1 * 5 = 5$
total memory bandwidth of all instances (per-instance * #instances)	$20 * 3 = 60$	$5 * 1 = 5$
total required bandwidth	$60 + 5 = 65$	
baseline GB/s per thread (total GB/s #threads)	$32/8 = 4$	
allocated baseline per thread (min(peak,baseline))	$min(20, 4) = 4$	$min(1, 4) = 1$
allocated node GB/s ($\sum$ #apps * #threads * GB/s per thread)	$3 * 1 * 4 + 1 * 5 * 1 = 17$	
remaining node GB/s	$32 - 17 = 15$	
still required GB/s per thread (peak - allocated)	$20 - 4 = 16$	$1 - 1 = 0$
still required GB/s	$3 * 1 * 16 + 1 * 5 * 0 = 48$	
remainder given to a thread (remaining node GB/s / unsatisfied threads)	$15/(3 * 1) = 5$	
total allocated to each thread (baseline + split remainder)	$4 + 5 = 9$	$1 + 0 = 1$
GFLOPS per thread (allocated GB/s * AI)	$9 * 0.5 = 4.5$	$1 * 10 = 10$
GFLOPS per application (#threads * per-thread)	$1 * 4.5 = 4.5$	$5 * 10 = 50$
total GFLOPS per node	$3 * 4.5 + 1 * 50 = 63.5$	
total GFLOPS	$4 * 63.5 = 254$	

Modeled scenarios



- different thread allocations
- memory bound and compute bound applications
- optimal layout depends on the actual AI of the applications
  - each of a), b), and c) can be a "winner"

Experimental evaluation

- on real hardware
  - 4x 20-core Skylake
- simple simulation of an application with fixed arithmetic intensity
- good match with model
  - less accuracy in presence of communication across NUMA nodes

scenario	model GFLOPS	real GFLOPS
uneven thr. (1,1,1,17)	23.20	22.82
even thr. (5,5,5,5)	18.12	18.14
one app per node	15.18	15.28
NUMA-bad cross-node	13.98	13.25
NUMA-bad on-node	15.18	14.52

Conclusion and future work

- cooperating applications are a great opportunity for dynamic runtime systems
  - the OS is a strong opponent to beat
  - fine grained resource sharing and synchronization possible among multiple processes (earlier work)
- NUMA is important
  - no “one size fits all”
- interesting insights even from simple models

- future work
  - put all this into practice
  - work smoothly across process boundaries
  - work smoothly with different runtime systems
    - task-based
  - work out the practical details
    - non-worker threads, external CPU load, …