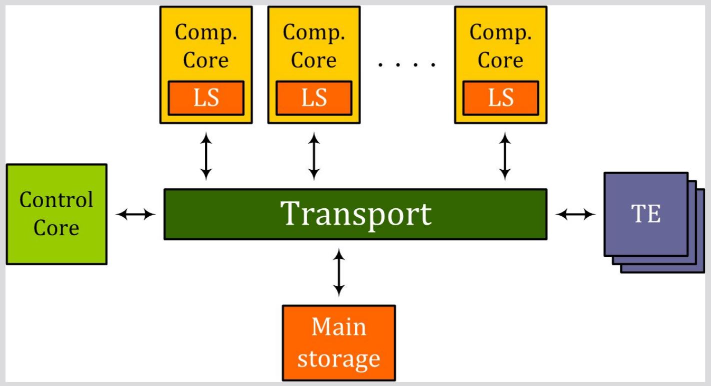
Adaptive libraries for multicore architectures with explicitly-managed memory hierarchies

Konstantin Nedovodeev,
Research Engineer at
the Institute for High-Performance Computer and Network Technologies

Key architectural features

Embedded MPSoC's with an explicitly-managed memory hierarhy (EMMA) posess:



- **three different types of cores**, namely:
    - control core(s);
    - "number-crunching" cores;
    - transfer engines (TE).
- each computational core has its **"private"** small sized **local store** (LS);
- there is a **big main storage** (RAM);
- all **inter-memory** transfers **ought to be managed by TEs** (hence **"explicitly-managed memory"** term).

Examples of such MPSoCs:

TI OMAP, TI DaVinci, IBM Cell, Atmel Diopsis, Broadcom mediaDSP, Elvees "Multicore" (Russia)

Programming issues

- **workload distribution** among computational cores;
- information **transfers distribution** among different channels:
- trying to **reuse data in the local store** (locality-awareness);
- trying to **use LS <-> LS** (bypassing) as much as possible;
- using multi-buffering to **hide memory latency**;
- local **memory allocation without fragmentation**;
- managing **synchronization** of parallel processes;
- **avoiding WaW, WaR dependencies** by allocating temporary store in common memory (results renaming).

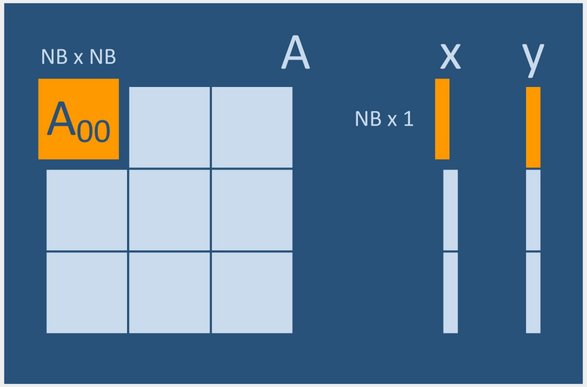
Tiled algorithms

We concentrate on a **high-performance tiled algorithm** construction.

Such algorithms are used in the **BLAS** library, which the **LAPACK** library is based on.

An example of a task for tiled algorithm construction is the matrix-vector product $y' = \alpha A x + \beta y$ (BLAS):

foreach i in (0..N')
  $y_i = \alpha A_{i0} x_0 + \beta y_i$
  foreach j in (1..N')
    $y_i = \alpha A_{ij} x_j + y_i$

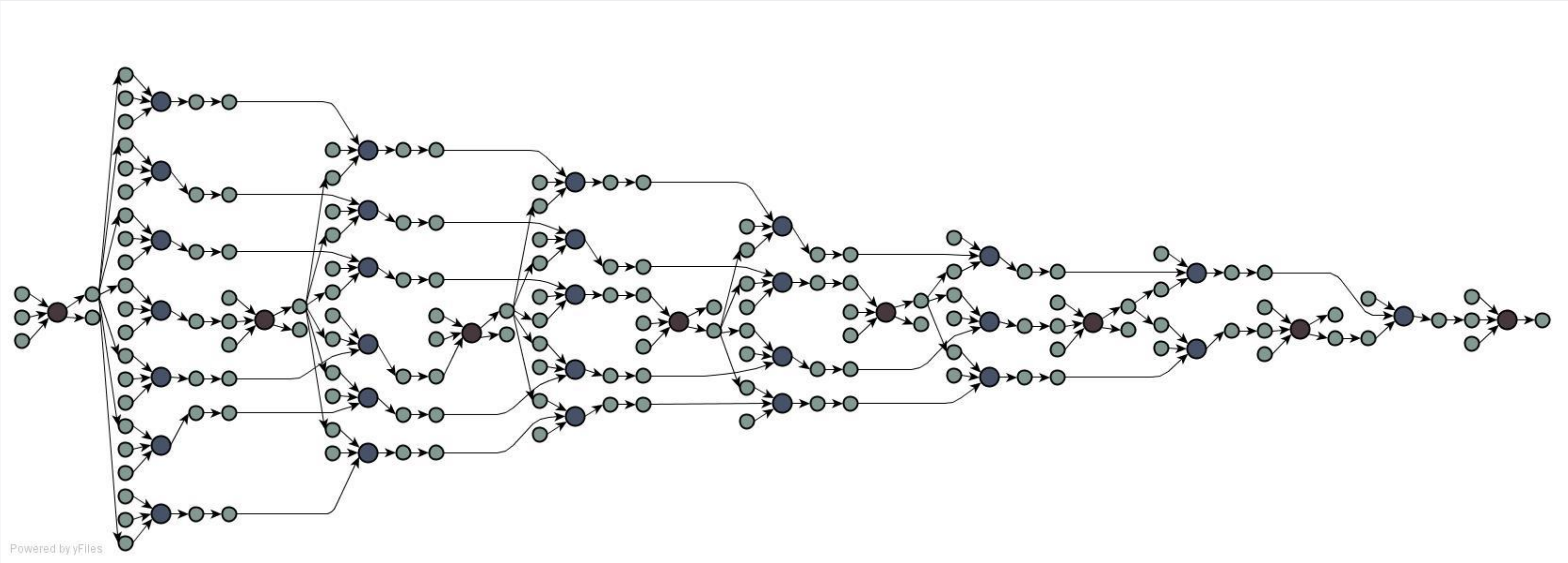


The tile is a rectangular dense submatrix.

where $A \in R^{NxN}$, $x,y \in R^N$, $\alpha, \beta \in R$, NB – blocking factor, N' = ceil(N / NB).

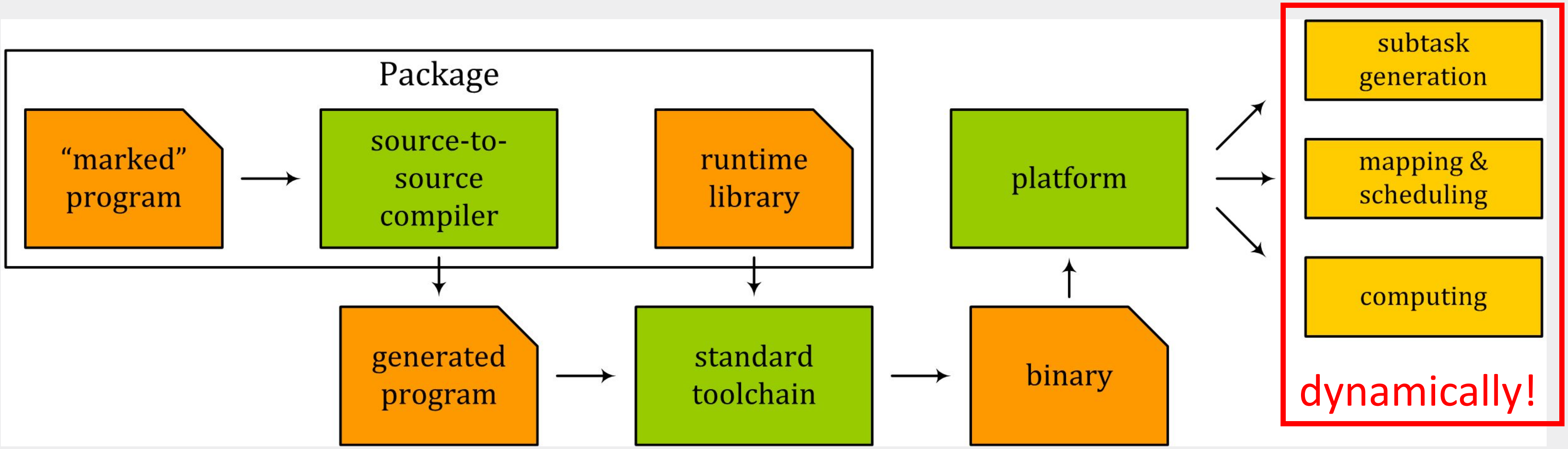
Program as a coarse-grained dataflow graph

Each program could be represented as a macro-flow graph.



Bigger nodes represent microkernel calls performed by the computational cores, while smaller ones represent tile transfers between the main storage and LS.

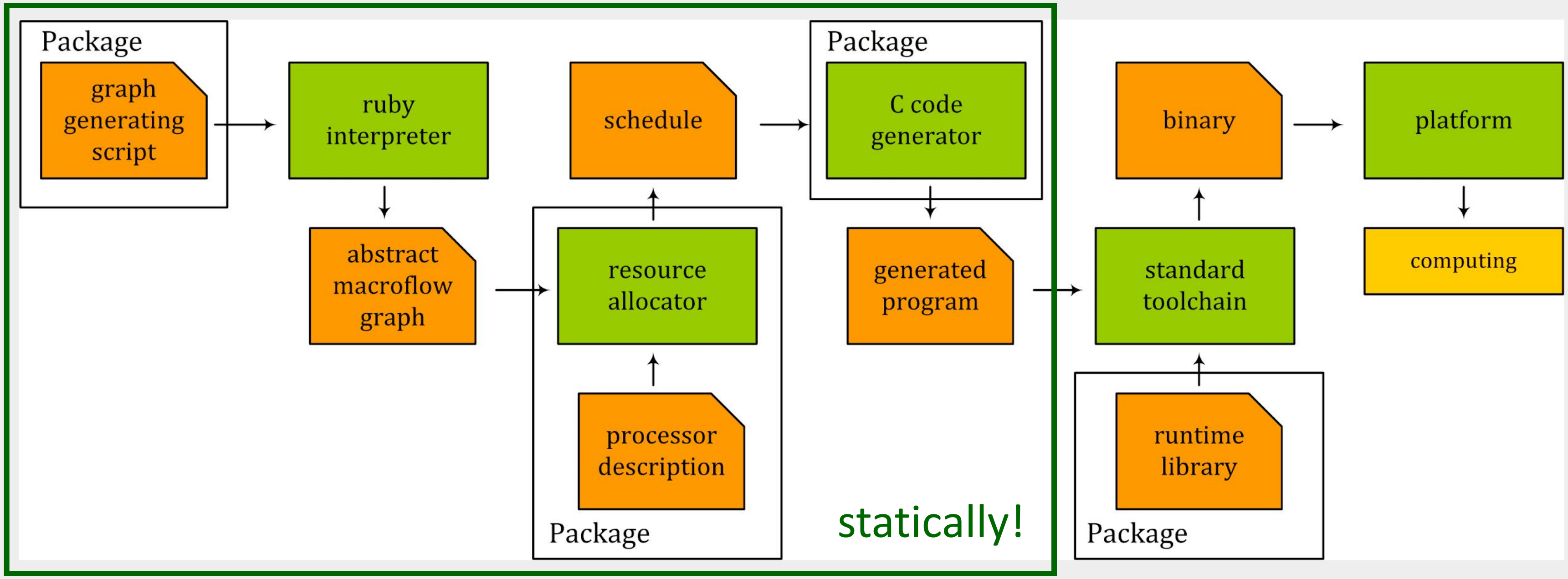
Existing toolchains



Existing toolchains (Cilk, StarS) **make scheduling decisions at runtime**. The runtime manages tasks (tile processing), distribute the workload, try to do its best in memory reuse, etc.

While being flexible, it **lacks unification** for EMMA platforms and **leads to a significant penalty** for small to medium-sized problems.

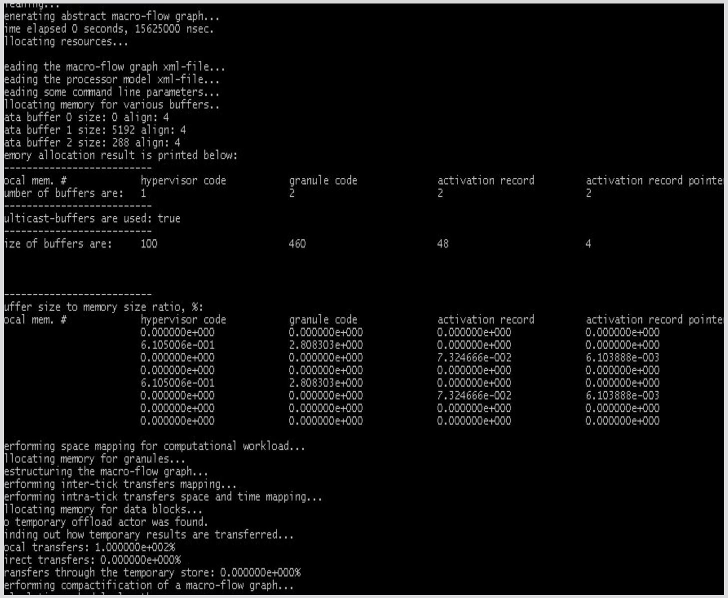
Proposed toolchain



Our approach **moves "decision-making" to compile-time**, reducing the overhead level. It becomes possible, because, the computational process does not depend on particular data values.

Standalone graph-generating scripts and processor model description make the library both **portable and extensible**.

How does it feel?



User:

1. Wants to **generate** a parallel **program**.

Runs single command, e.g.:

sampl_make_src.bat strsv 70 35 mc0226 2

and gets the source files.

Support engineer:

1. Wants **to port** the library.

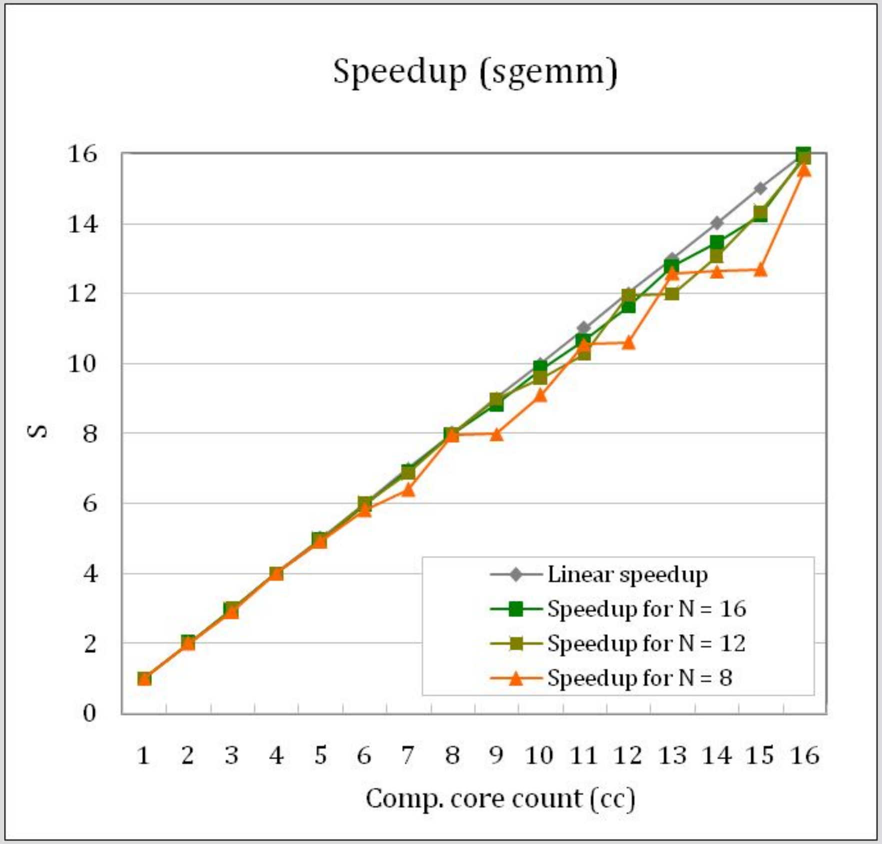
Writes a new version of the runtime-library (200-300 LOC).

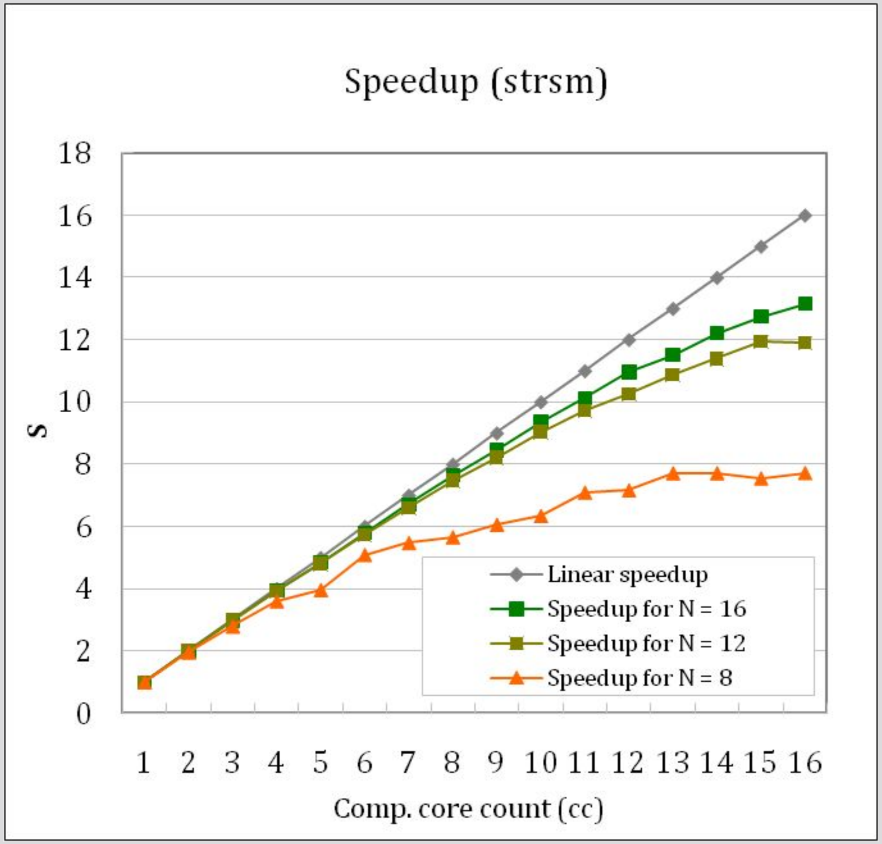
2. Wants **to add a new program**.

Writes the Ruby script (400-500 boilerplate LOC) (DSL?).

Writes microkernels for computational cores (~100 ASM LOC per mk).

How fast is it?

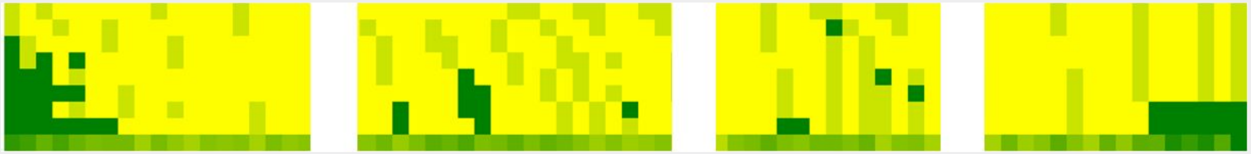




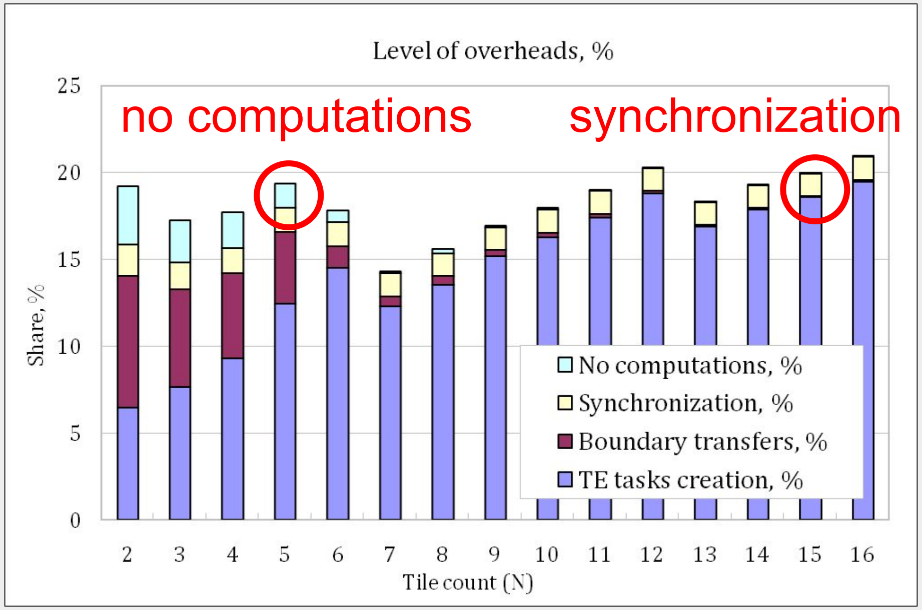
Scales almost linearly up to 16 cores of a synthetic multicore processor (Matrix size = N · NB).

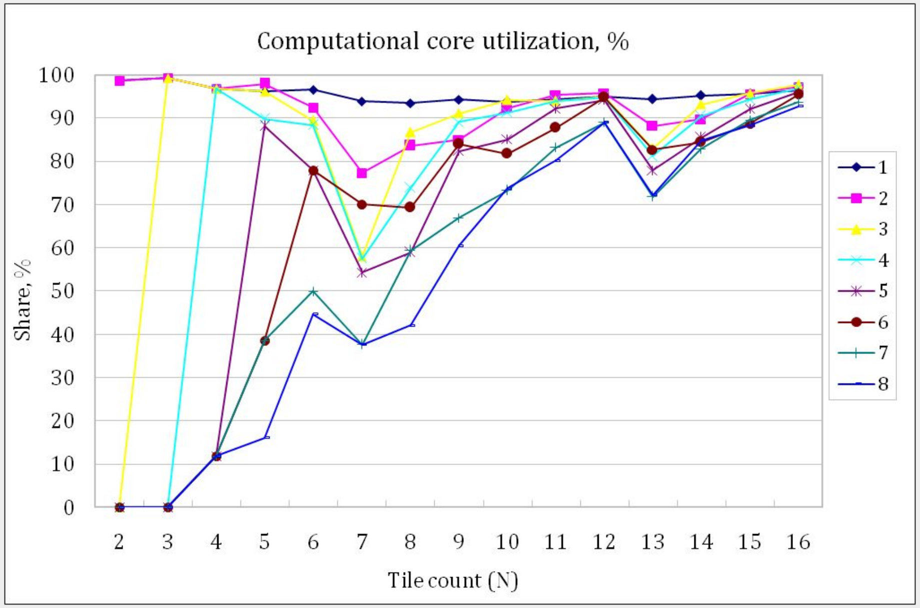
SGEMM – matrix multiplication, STRSM – triangular solve with multiple right-hand sides.

How fast is it (continued)?



The heatmap of an STRSM program schedule (cc = 8).





< 20% of the time the control core makes TE tasks (overlaps with computations),

< 2% it spends for synchronization. **Computational cores work almost all the time**!