Scalable Matrix Architecture (Fundamentals and Vector-Matrix Facility)

José Moreira
IBM Corporation

Abel Bernabeu
Esperanto

Comments on the state-of-the-art for matrix computing

- Most high-performance matrix computations rely on the following kernel:

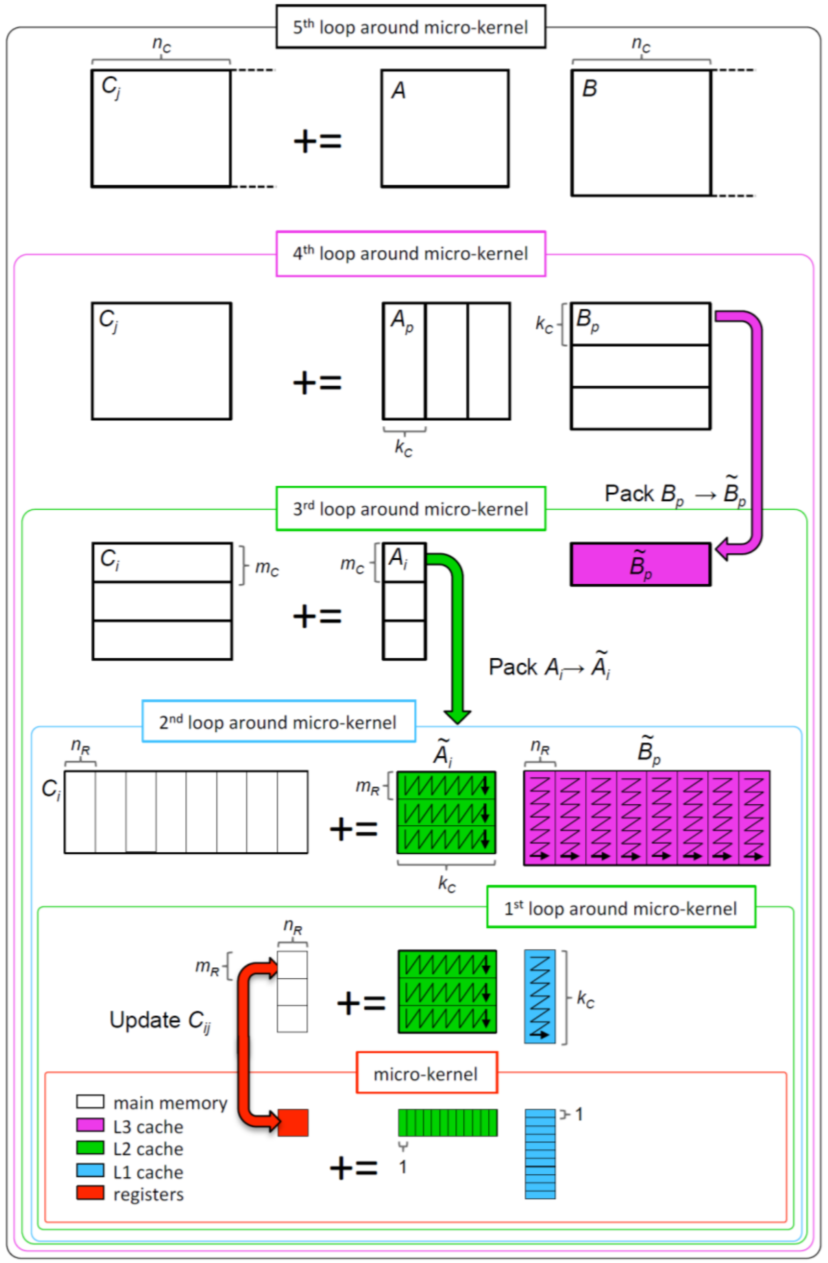
$$\boldsymbol{C}_{m\times n} = \boldsymbol{A}_{m\times K} \times \boldsymbol{B}_{K\times n}$$

$\boldsymbol{C}_{m\times n}$	is a register-resident matrix, with $m \approx n$, and
$\boldsymbol{A}_{m\times K}$ and $\boldsymbol{B}_{K\times n}$	are streamed from memory, with $m \ll K$ and $n \ll K$

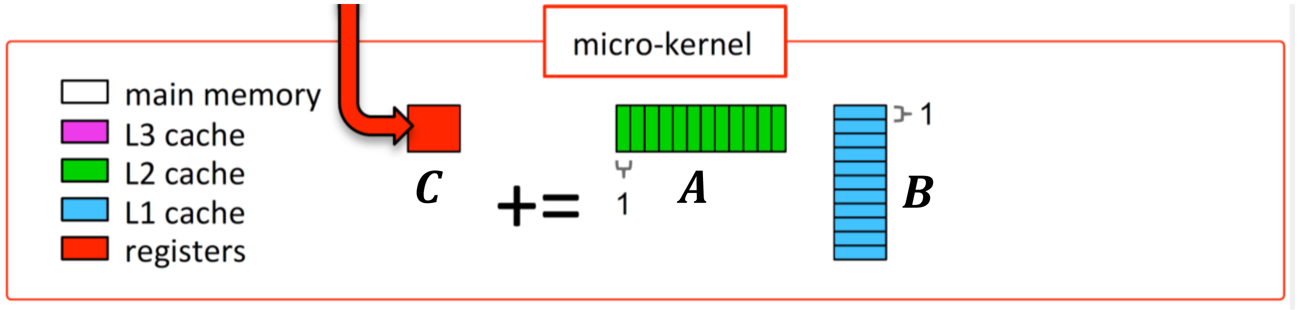
- Matrix $\boldsymbol{A}_{m\times K}$ is organized as a vector of K elements, each element a column vector of m elements of $\boldsymbol{A} = [\boldsymbol{A}^0 \quad \boldsymbol{A}^1 \quad \cdots \quad \boldsymbol{A}^{K-1}]$
- Matrix $\boldsymbol{B}_{K\times n}$ is organized as a vector of K elements, each element a row of n elements of $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_0 \\ \boldsymbol{B}_1 \\ \vdots \\ \boldsymbol{B}_{K-1} \end{bmatrix}$
- We will consider only the case of deterministic values of $\boldsymbol{C}$, as produced by the following algorithm:

$$\boldsymbol{C} \leftarrow 0; \text{for } \left(k = [0, K)\right) \boldsymbol{C} \leftarrow \boldsymbol{A}^k \times \boldsymbol{B}_k + \boldsymbol{C}$$

Anatomy of a high-performance matrix multiply



5th loop around micro-kernel
n_C
C_j
+=
A
B
4th loop around micro-kernel
A_p
k_C
B_p
Pack $B_p \rightarrow \tilde{B}_p$
3rd loop around micro-kernel
C_i
m_C
A_i
$\tilde{B}_p$
Pack $A_i \rightarrow \tilde{A}_i$
2nd loop around micro-kernel
n_R
m_R
$\tilde{A}_i$
1st loop around micro-kernel
Update C_{ij}
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers
1



Performance bounds

- Let Δ be the latency (in cycles) of an elemental multiply-add operation
- Computing each element of $\boldsymbol{C}$, as described above, requires evaluating a dependence chain of K multiply-adds, and therefore takes time $K\Delta$

$$\boldsymbol{C} \leftarrow \underbrace{\boldsymbol{A}^k \times \boldsymbol{B}_k + \boldsymbol{C}}_{\Delta}$$

- The computation of $\boldsymbol{C}$ requires mnK multiply-adds
- The maximum computation rate that can be sustained is

$$R = \frac{mnK}{K\Delta} = {}^{mn}\!/_{\Delta} \quad \text{madds/cycle}$$

- This sets an upper bound on performance, dictated by the size of the $\boldsymbol{C}$ panel that can be kept in registers (an architectural parameter) and the latency of a multiply-add (a design/technology parameter) – *Note*: integer arithmetic is more forgiving
- For modern server processor and 32-bit floating-point elements, a reasonable value of Δ is 4 cycles – the range $\Delta \in [2,8]$ should cover most cases

Some configurations (just to get a feel)

- Let the architected space consist of 32 vector registers of size L words (32-bit)
  - Then, $mn \leq 32L$ and the maximum computation rate is $R = {}^{32L}/_{4} = 8L \text{ madds/cycle}$ (single-precision floating-point)
- The maximum computation rate of a vector-based matrix facility, using vector registers only, scales with L
- For $L = 4$, $R = 32 \text{ madds/cycle}$
- For $L = 16$, $R = 128 \text{ madds/cycle}$
- *Note*: In the above, we use all architected space for $\boldsymbol{C}$, but in a load/store architecture we need to reserve space for $\boldsymbol{A}$ and $\boldsymbol{B}$, so cut the bound in half
  - $mn \leq 16L$ and the maximum computation rate is $R = {}^{16L}/_{4} = 4L \text{ madds/cycle}$
  - For $L = 4$, $R = 16 \text{ madds/cycle}$
  - For $L = 16$, $R = 64 \text{ madds/cycle}$
- *Note*: $4L$ madds/cycle is the performance equivalent of 4 vector pipes operating concurrently

Conclusions

- The size of the architected register space imposes an upper bound on the performance of deterministic matrix multiply kernels
- If we use the architected scalable vector register space ($32L$ words) to hold the panel of $\boldsymbol{C}$ our performance upper bound, for a reasonable choice of latency parameters, is $8L$ madds/cycle – a more realistic value is $4L$ madds/cycle (single-precision floating-point)
- We advocate to pursue the vector register-based matrix extensions first, starting with three possible options:
  - Option A : 1 matrix/1 vector register ($R = 4L$ madds/cycle with 4 matrix pipes)
  - Option B : 1 matrix/n vector registers ($R = 4L$ madds/cycle with 1 matrix pipe)
  - Option C : n matrices/1 vector register ($R = 4L$ madds/cycle with ${}^{4}/_{\lambda}$ matrix pipes)