Computational Science on Many-Core Architectures

360.252

Josef Weinbub
Karl Rupp

Institute for Microelectronics, TU Wien
http://www.iue.tuwien.ac.at/

Zoom Channel 621 2711 2607
Wednesday, November 8, 2023

Kernel Fusion - Multiple Dot Products

Gram-Schmidt method

- Given orthonormal basis $\{v_1, v_2, \ldots, v_N\}$, augment by w
- $w \leftarrow w - \langle w, v_i \rangle v_i$
- $w \leftarrow w / \|w\|$
- Add w to basis

Multiple inner products $\langle w, v_i \rangle$

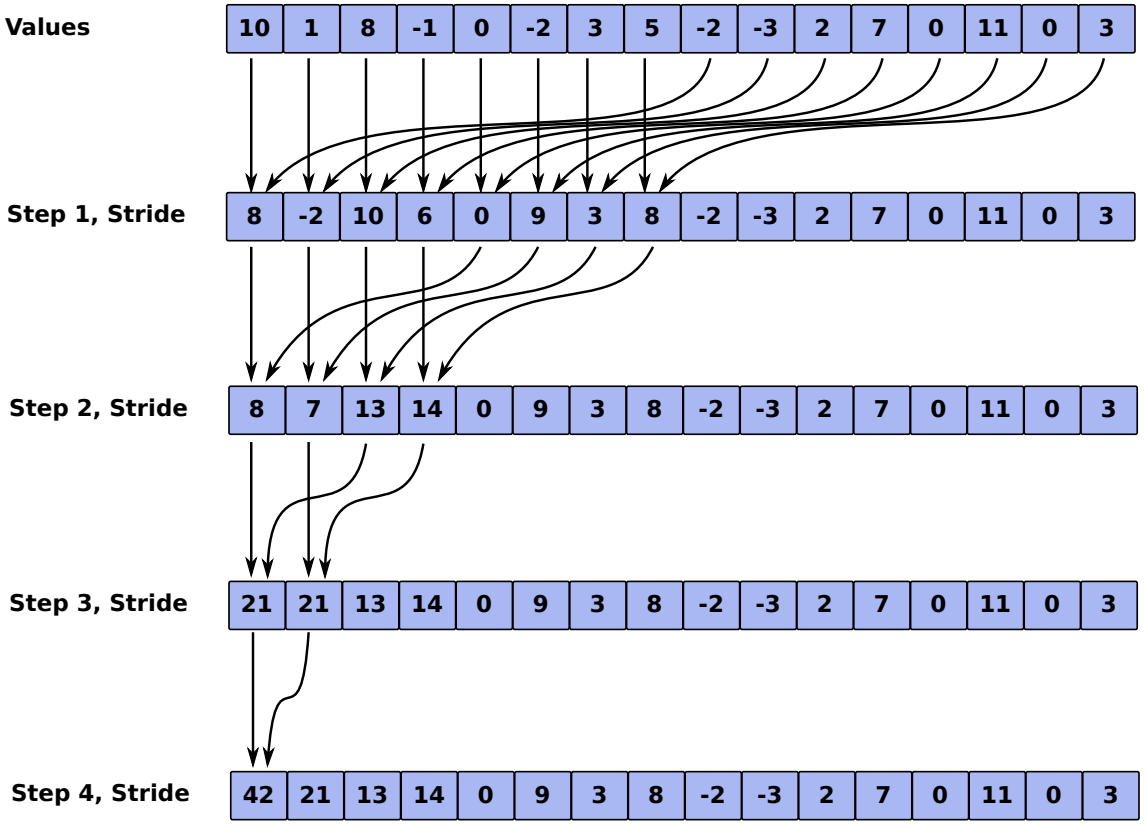
- Performance critical (global reductions)
- Reuse of w desirable

Implementation 1: dot

Goal: Compute $\alpha_i = \langle w, v_i \rangle$ for $i = 1, \ldots, N$ efficiently

Method 1: Iterated application of *dot()*

- Decompose into N separate inner products
- N calls to BLAS level 1 routine *ddot*
- Reductions: One per work group, one over results of work groups



Implementation 2: gemv

Goal: Compute $\alpha_i = \langle w, v_i \rangle$ for $i = 1, \ldots, N$ efficiently

Method 2: Pack vectors into matrix, use *gemv*

- Set $\mathbf{A} = \begin{pmatrix} v_1^{\mathrm{T}} \\ \vdots \\ v_N^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{N \times M}, N \ll M$
- Compute $\boldsymbol{\alpha} = \mathbf{A}\mathbf{x}$
- One BLAS level 2 *dgemv* call

Implementation 3: mdot

Method 3: Custom routine *mdot*

- Process $\alpha_i = \langle w, v_i \rangle$ in batches
- Batch sizes 1, 2, 3, 4, 8
- Load entries of w only once per batch
- Fit remaining inner products into largest batch possible
- Batch size 8: Only 12.5% overhead vs. arbitrary batch sizes

Implementation 3: mdot

Method 3: Custom routine *mdot*

- Process $\alpha_i = \langle w, v_i \rangle$ in batches
- Batch sizes 1, 2, 3, 4, 8
- Load entries of w only once per batch
- Fit remaining inner products into largest batch possible
- Batch size 8: Only 12.5% overhead vs. arbitrary batch sizes

Code sketch (Batch size 4)

```
for (size_t i=thread_id; i<M; i += threads_per_group)
{
  double val_w = w[i];
  alpha_1 += val_w * v1[i];
  alpha_2 += val_w * v2[i];
  alpha_3 += val_w * v3[i];
  alpha_4 += val_w * v4[i];
}
```