Mei Wen, Da-fei Huang, Chang-qing Xun, Dong Chen, 2015. Improving performance portability for GPU-specific OpenCL kernels on multi-core/ many-core CPUs by analysis-based transformations. *Frontiers of Information Technology & Electronic Engineering*, **16**(11):899-916 . [doi:10.1631/FITEE.1500032]

Improving performance portability for GPU-specific OpenCL kernels on multi-core/many-core CPUs by analysis-based transformations

Key words: OpenCL, Performance portability, Multi-core/many-core CPU, Analysis-based transformation

Contact: Da-fei Huang
E-mail: huangdafei1012@163.com
ORCID: http://orcid.org/0000-0001-6617-7608

Introduction

- OpenCL provides functional portability, but does not guarantee performance portability.
- The majority of existing OpenCL programs are GPU-specific, which typically cannot achieve good performance on CPUs.
- To improve performance portability, a new kind of array-access descriptors is derived from kernels, which is then used to:
  - 1) remove unwanted local-memory arrays together with obsolete barriers;
  - 2) perform vectorization and locality re-exploitation.

A linear descriptor of array access

$$\begin{cases} f_{\boldsymbol{AS}}^{\text{write}} = \text{Lid}.x + \text{Lid}.y \times \text{BLOCK\_SIZE}, \\ \text{Constraint}_{\boldsymbol{AS}}^{\text{write}} = \{\text{Lid}.x \geq 0;\ \text{Lid}.x < \text{BLOCK\_SIZE};\ \text{Lid}.y \geq 0;\ \text{Lid}.y < \text{BLOCK\_SIZE}\}, \end{cases}$$

```
AS[Lid.x + Lid.y * BLOCK_SIZE] = A[a + uiWA * Lid.y + Lid.x];
```

$$\begin{cases} f_{\boldsymbol{A}}^{\text{read}} = (\text{uiWA} \times \text{BLOCK\_SIZE} \times \text{Gid}.y + \text{BLOCK\_SIZE} \times \text{Iter}_a) + \text{uiWA} \times \text{Lid}.y + \text{Lid}.x, \\ \text{Constraint}_{\boldsymbol{A}}^{\text{read}} = \{\text{Iter}_a \geq 0\ ;\ \ \text{Iter}_a < \text{uiWA}/\text{BLOCK\_SIZE};\ \text{Gid}.y \geq 0;\ \text{Gid}.y < \text{GLOBAL\_SIZE}; \\ \qquad \text{Lid}.x \geq 0;\ \text{Lid}.x < \text{BLOCK\_SIZE};\ \text{Lid}.y \geq 0;\ \text{Lid}.y < \text{BLOCK\_SIZE}\}, \end{cases}$$

```
Csub += AS[k + Lid.y*BLOCK_SIZE] * BS[Lid.x + k*BLOCK_SIZE];
```

$$\begin{cases} f_{\boldsymbol{AS}}^{\text{read}} = \text{Iter}_k + \text{Lid}.y \times \text{BLOCK\_SIZE}, \\ \text{Constraint}_{\boldsymbol{AS}}^{\text{read}} = \{\text{Iter}_k \geq 0;\ \ \text{Iter}_k < \text{BLOCK\_SIZE};\ \text{Lid}.y \geq 0;\ \text{Lid}.y < \text{BLOCK\_SIZE}\}. \end{cases}$$

Analysis-based coalescing

- Eliminating unnecessary local-memory arrays

$$\begin{cases} f_{\boldsymbol{AS}}^{\text{write}} = \text{Lid}.x + \text{Lid}.y \times \text{BLOCK\_SIZE}, \\ \text{Constraint}_{\boldsymbol{AS}}^{\text{write}} = \\ \{\text{Lid}.x \geq 0;\ \text{Lid}.x < \text{BLOCK\_SIZE};\ \text{Lid}.y \geq 0; \\ \ \text{Lid}.y < \text{BLOCK\_SIZE}\}, \end{cases} \tag{1}$$

$$\begin{cases} f_{\boldsymbol{AS}}^{\text{read}} = \text{Iter}_k + \text{Lid}.y \times \text{BLOCK\_SIZE}, \\ \text{Constraint}_{\boldsymbol{AS}}^{\text{read}} = \\ \{\text{Iter}_k \geq 0;\ \ \text{Iter}_k < \text{BLOCK\_SIZE};\ \text{Lid}.y \geq 0; \\ \ \text{Lid}.y < \text{BLOCK\_SIZE}\}. \end{cases} \tag{2}$$

If write-read descriptor pair becomes identical via variable substitution

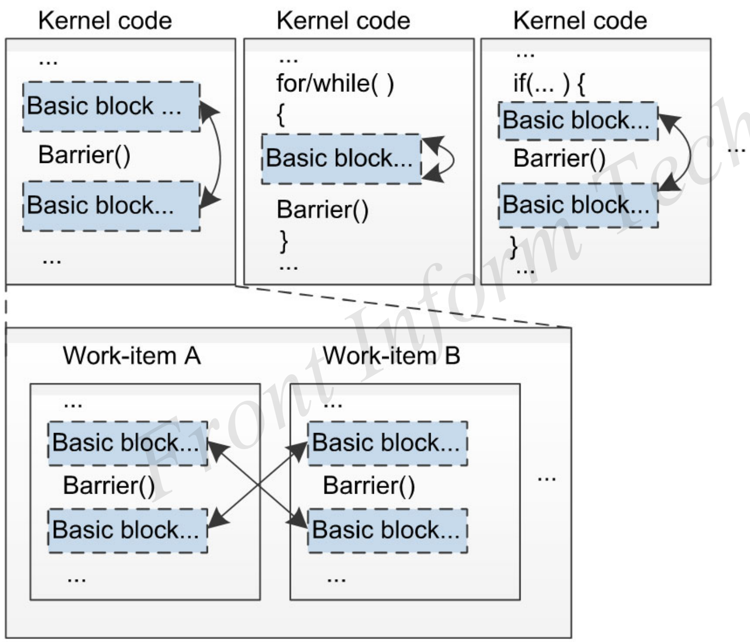
$$\begin{aligned} & f_{\boldsymbol{AS}}^{\text{read}} = \text{Iter}_k + \text{Lid}.y \times \text{BLOCK\_SIZE} \\ & \Rightarrow \\ & f_{\boldsymbol{A}}^{\text{read}} = (\text{uiWA} \times \text{BLOCK\_SIZE} \times \text{Gid}.y + \\ & \text{BLOCK\_SIZE} \times \text{Iter}_a) + \text{uiWA} \times \text{Lid}.y + \text{Iter}_k. \end{aligned}$$

Local memory read can be transformed to direct global memory read

Analysis-based coalescing

- Eliminating barriers via dependence analysis

Dependence analysis scope



Dependence analysis equation

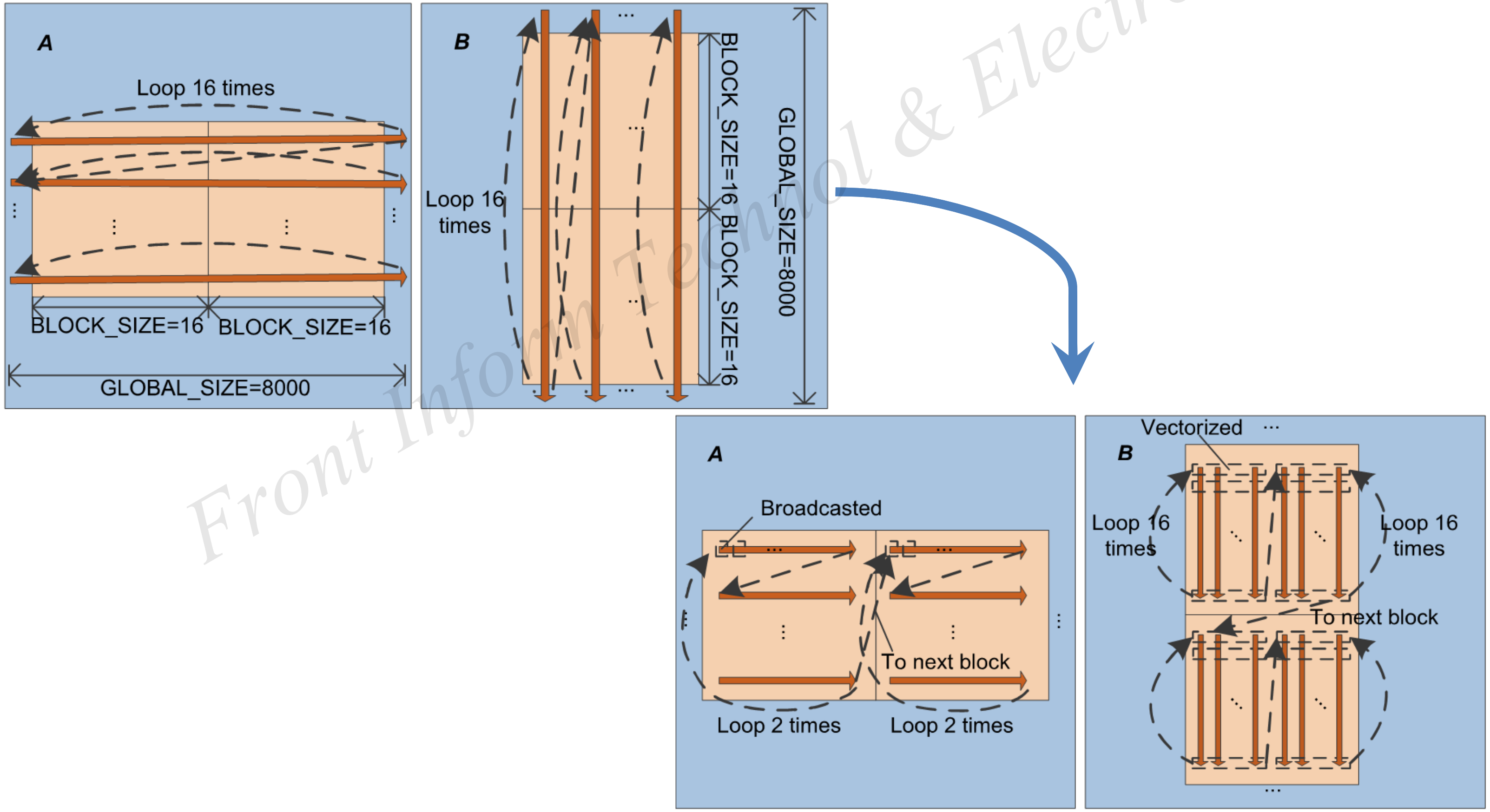
$$\begin{cases} f_1 = \overrightarrow{\mathbf{Coe}_1} \cdot \overrightarrow{\mathbf{Var}_1}^{\mathrm{T}} + \mathrm{Const}, \\ \mathrm{Constraint}_1 \quad \overrightarrow{\mathbf{Var}_1} = (..., \mathrm{Lid}.z, \mathrm{Lid}.y, \mathrm{Lid}.x), \end{cases}$$

$$\begin{cases} f_2 = \overrightarrow{\mathbf{Coe}_2} \cdot \overrightarrow{\mathbf{Var}_2}^{\mathrm{T}} + \mathrm{Const}, \\ \mathrm{Constraint}_2 \quad \overrightarrow{\mathbf{Var}_2} = (..., \mathrm{Lid}.z', \mathrm{Lid}.y', \mathrm{Lid}.x'), \end{cases}$$

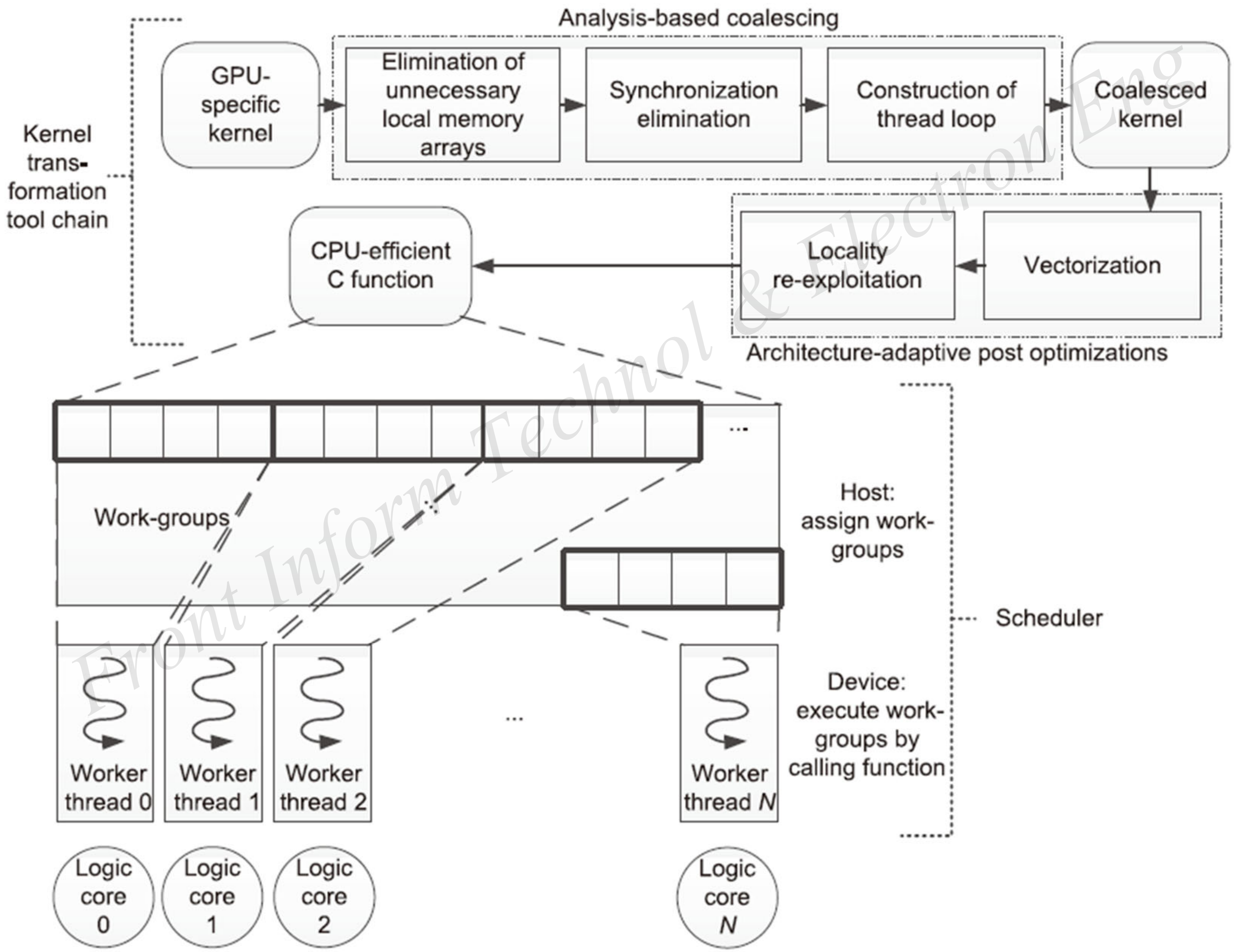
$$\Rightarrow \begin{cases} f_1 = f_2, \\ \mathrm{Constraint}_1, \\ \mathrm{Constraint}_2. \end{cases}$$

Architecture-adaptive post optimizations

- Vectorization
- Data locality re-exploitation



Tool chain and supporting scheduler



Performance evaluation

- Kernel execution time

Table 2 Performance comparison with Intel OpenCL implementation and OpenMP

Kernel name	Performance ratio (execution time)					
	CPU+IntelOCL	CPU+OurOCL	CPU+OMP	MIC+IntelOCL	MIC+OurOCL	MIC+OMP
oclMatrixMul	1 (23.30 s)	3.02 (7.71 s)	0.37 (62.98 s)	1.94 (12.03 s)	3.93 (5.93 s)	3.74 (6.23 s)
oclFDTD3d	1 (0.80 s)	6.15 (0.13 s)	2.16 (0.37 s)	2.22 (0.36 s)	5.71 (0.14 s)	4.21 (0.19 s)
Stencil2D	1 (20.65 s)	2.53 (8.16 s)	1.16 (17.80 s)	1.83 (11.26 s)	2.42 (8.53 s)	1.95 (10.59 s)
oclDCT8x8	1 (75.96 ms)	3.42 (22.20 ms)	2.27 (33.46 ms)	1.43 (53.22 ms)	4.18 (18.19 ms)	4.52 (16.81 ms)
oclNbody	1 (10.63 s)	1.20 (8.82 s)	0.74 (14.36 s)	1.13 (9.44 s)	1.24 (8.59 s)	1.38 (7.70 s)
NaiveMatrixMul	1 (258.16 s)	33.44 (7.72 s)	4.10 (62.98 s)	4.55 (56.73 s)	43.76 (5.90 s)	41.44 (6.23 s)

Performance evaluation

- Performance improvement due to each step

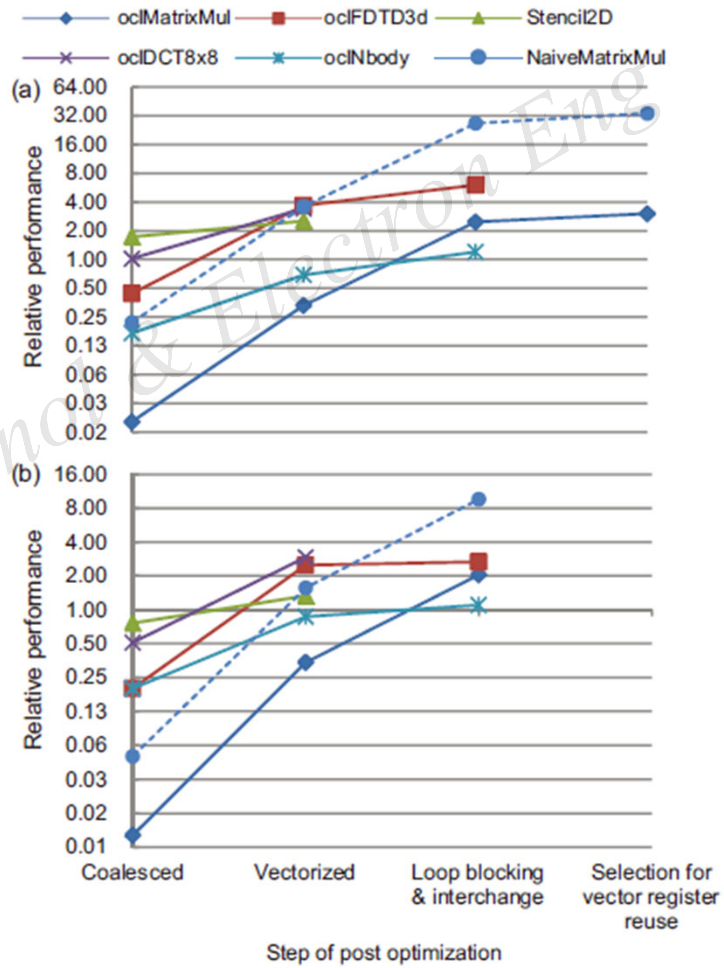


Fig. 12 Performance improvement compared to the relevant original GPU-specific kernels due to each step within post optimization: (a) CPU; (b) MIC

Conclusions

- To improve the performance portability of OpenCL kernels from GPUs to CPUs, this paper presents a novel and effective transformation methodology based on array access analysis.

- The proposed methodology achieves high speedup by alleviating two particular problems that have not been well addressed:
  - The potential side-effects induced by using local-memory arrays on CPUs;
  - The possibility of poor data locality caused by neglecting or blindly inheriting locality embedded in the original GPU-specific kernels.