PetaBricks: A Language and Compiler for Algorithmic Choice [1]

Authors: Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, Saman Amarasinghe.
MIT CSAIL

Presented by Umer Hasan (suh25)

Michelmas 2025

UNIVERSITY OF CAMBRIDGE

Motivation and Background

What is it?

It's a programming language and a compiler (Petabricks -> C++) and a run-time library implemented in C++.

UNIVERSITY OF CAMBRIDGE

Motivation and Background

What is it?

It's a programming language and a compiler (Petabricks -> C++) and a run-time library implemented in C++.

Why do we need it?

We need **algorithmic choice**: for a given task e.g. sorting, no single algorithm is optimal across all contexts (e.g. `std::sort` has fixed cutoff merge -> insertion sort).

Motivation and Background

What is it?

It's a programming language and a compiler (Petabricks -> C++) and a run-time library implemented in C++.

Why do we need it?

We need **algorithmic choice**: for a given task e.g. sorting, no single algorithm is optimal across all contexts (e.g. `std::sort` has fixed cutoff merge -> insertion sort).
We need **simplicity**:

- Handle switching algorithms based on params and input data (avoids obfuscated code and manual tuning). Replace ATLAS [5] / FFTW [4] (C tuning libraries).

Motivation and Background

What is it?

It's a programming language and a compiler (Petabricks -> C++) and a run-time library implemented in C++.

Why do we need it?

We need **algorithmic choice**: for a given task e.g. sorting, no single algorithm is optimal across all contexts (e.g. `std::sort` has fixed cutoff merge -> insertion sort).
We need **simplicity**:

- Handle switching algorithms based on params and input data (avoids obfuscated code and manual tuning). Replace ATLAS [5] / FFTW [4] (C tuning libraries).
- 1 optimal implementation ported to different machines by re-compiling and autotuning (x86 vs Sun Niagara).

OF CAMBRIDGE

Features

- Algorithmic choice through 1 `transform` with multiple `rules`.

```
1  transform MatrixMultiply
2  from A[c,h], B[w,c]
3  to AB[w,h]
4  {
5    // Base case, compute a single element
6    to(AB.cell(x,y) out)
7    from(A.row(y) a, B.column(x) b) {
8      out = dot(a,b);
9    }
10
11   // Recursively decompose in c
12   to(AB ab)
13   from(A.region(0,    0, c/2, h  ) a1,
14        A.region(c/2, 0, c,    h  ) a2,
15        B.region(0,    0, w,    c/2) b1,
16        B.region(0, c/2, w,    c  ) b2) {
17     ab = MatrixAdd(MatrixMultiply(a1, b1),
18                    MatrixMultiply(a2, b2));
19   }
20
21   // Recursively decompose in w
22   to(AB.region(0,    0, w/2, h  ) ab1,
23      AB.region(w/2, 0, w,    h  ) ab2)
24   from( A a,
25         B.region(0,    0, w/2, c  ) b1,
26         B.region(w/2, 0, w,    c  ) b2) {
27     ab1 = MatrixMultiply(a, b1);
28     ab2 = MatrixMultiply(a, b2);
29   }
30
31   // Recursively decompose in h
32   to(AB.region(0, 0,    w, h/2) ab1,
33      AB.region(0, h/2, w, h  ) ab2)
34   from(A.region(0,    0, c,    h/2) a1,
35        A.region(0, h/2, c,    h  ) a2,
36        B b) {
37     ab1=MatrixMultiply(a1, b);
38     ab2=MatrixMultiply(a2, b);
39   }
40 }
```

Figure 1: MatrixMultiply PetaBricks source code

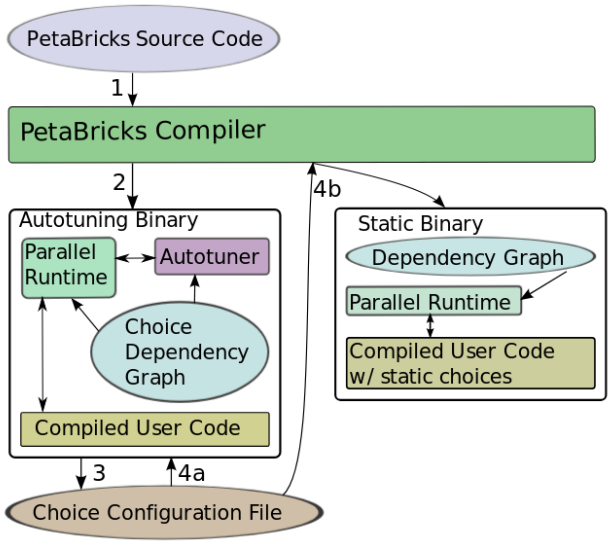

Figure 2: Interactions between compiler and output binaries.

UNIVERSITY OF CAMBRIDGE

Features

- Algorithmic choice through 1 `transform` with multiple `rules`.
- The compilation uses a Choice Dependency Graph (CDG), nodes = transforms/rules, edges = (data) dependencies.

```
1  transform MatrixMultiply
2  from A[c,h], B[w,c]
3  to AB[w,h]
4  {
5    // Base case, compute a single element
6    to(AB.cell(x,y) out)
7    from(A.row(y) a, B.column(x) b) {
8      out = dot(a,b);
9    }
10
11   // Recursively decompose in c
12   to(AB ab)
13   from(A.region(0,    0, c/2, h   ) a1,
14        A.region(c/2, 0, c,    h   ) a2,
15        B.region(0,    0, w,    c/2) b1,
16        B.region(0, c/2, w,    c   ) b2) {
17     ab = MatrixAdd(MatrixMultiply(a1, b1),
18                    MatrixMultiply(a2, b2));
19   }
20
21   // Recursively decompose in w
22   to(AB.region(0,    0, w/2, h   ) ab1,
23      AB.region(w/2, 0, w,    h   ) ab2)
24   from( A a,
25         B.region(0,    0, w/2, c   ) b1,
26         B.region(w/2, 0, w,    c   ) b2) {
27     ab1 = MatrixMultiply(a, b1);
28     ab2 = MatrixMultiply(a, b2);
29   }
30
31   // Recursively decompose in h
32   to(AB.region(0, 0,    w, h/2) ab1,
33      AB.region(0, h/2, w, h   ) ab2)
34   from(A.region(0,    0, c,    h/2) a1,
35        A.region(0, h/2, c,    h   ) a2,
36        B b) {
37     ab1=MatrixMultiply(a1, b);
38     ab2=MatrixMultiply(a2, b);
39   }
40 }
```

Figure 1: MatrixMultiply PetaBricks source code

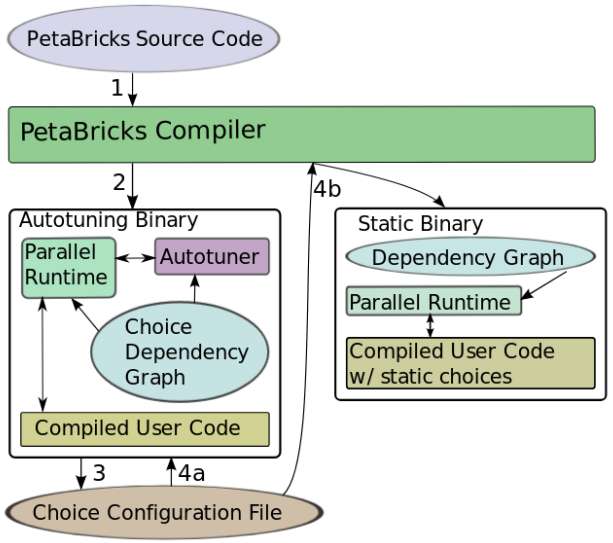

Figure 2: Interactions between compiler and output binaries.

UNIVERSITY OF CAMBRIDGE

Features

- Algorithmic choice through 1 `transform` with multiple `rules`.
- The compilation uses a Choice Dependency Graph (CDG), nodes = transforms/rules, edges = (data) dependencies.
- Accuracy choice with iterative and recursive methods.

```
1  transform MatrixMultiply
2  from A[c,h], B[w,c]
3  to AB[w,h]
4  {
5    // Base case, compute a single element
6    to(AB.cell(x,y) out)
7    from(A.row(y) a, B.column(x) b) {
8      out = dot(a,b);
9    }
10
11   // Recursively decompose in c
12   to(AB ab)
13   from(A.region(0,    0, c/2, h   ) a1,
14        A.region(c/2, 0, c,    h   ) a2,
15        B.region(0,    0, w,    c/2) b1,
16        B.region(0, c/2, w,    c   ) b2) {
17     ab = MatrixAdd(MatrixMultiply(a1, b1),
18                    MatrixMultiply(a2, b2));
19   }
20
21   // Recursively decompose in w
22   to(AB.region(0,    0, w/2, h   ) ab1,
23      AB.region(w/2, 0, w,    h   ) ab2)
24   from( A a,
25         B.region(0,    0, w/2, c   ) b1,
26         B.region(w/2, 0, w,    c   ) b2) {
27     ab1 = MatrixMultiply(a, b1);
28     ab2 = MatrixMultiply(a, b2);
29   }
30
31   // Recursively decompose in h
32   to(AB.region(0, 0,    w, h/2) ab1,
33      AB.region(0, h/2, w, h   ) ab2)
34   from(A.region(0,    0, c,    h/2) a1,
35        A.region(0, h/2, c,    h   ) a2,
36        B b) {
37     ab1=MatrixMultiply(a1, b);
38     ab2=MatrixMultiply(a2, b);
39   }
40 }
```

Figure 1: MatrixMultiply PetaBricks source code

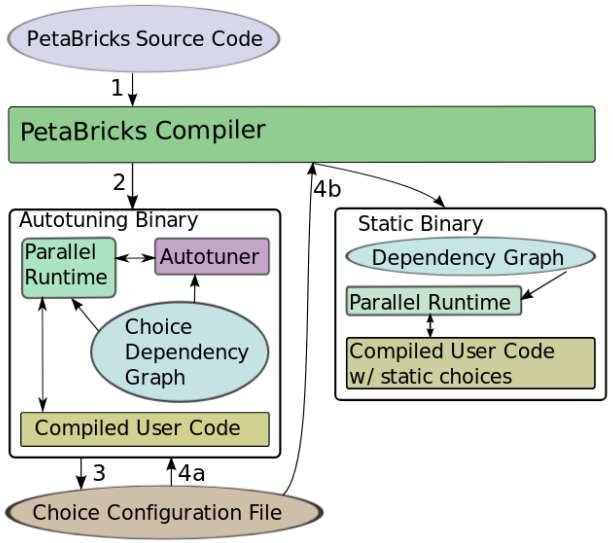

Figure 2: Interactions between compiler and output binaries.

UNIVERSITY OF CAMBRIDGE

Features

- Algorithmic choice through 1 `transform` with multiple `rules`.
- The compilation uses a Choice Dependency Graph (CDG), nodes = transforms/rules, edges = (data) dependencies.
- Accuracy choice with iterative and recursive methods.
- How does the autotuner select the best rule?

```
1  transform MatrixMultiply
2  from A[c,h], B[w,c]
3  to AB[w,h]
4  {
5    // Base case, compute a single element
6    to(AB.cell(x,y) out)
7    from(A.row(y) a, B.column(x) b) {
8      out = dot(a,b);
9    }
10
11   // Recursively decompose in c
12   to(AB ab)
13   from(A.region(0,    0, c/2, h   ) a1,
14        A.region(c/2, 0, c,    h   ) a2,
15        B.region(0,    0, w,    c/2) b1,
16        B.region(0, c/2, w,    c   ) b2) {
17     ab = MatrixAdd(MatrixMultiply(a1, b1),
18                    MatrixMultiply(a2, b2));
19   }
20
21   // Recursively decompose in w
22   to(AB.region(0,    0, w/2, h   ) ab1,
23      AB.region(w/2, 0, w,    h   ) ab2)
24   from( A a,
25         B.region(0,    0, w/2, c   ) b1,
26         B.region(w/2, 0, w,    c   ) b2) {
27     ab1 = MatrixMultiply(a, b1);
28     ab2 = MatrixMultiply(a, b2);
29   }
30
31   // Recursively decompose in h
32   to(AB.region(0, 0,    w, h/2) ab1,
33      AB.region(0, h/2, w, h   ) ab2)
34   from(A.region(0,    0, c,    h/2) a1,
35        A.region(0, h/2, c,    h   ) a2,
36        B b) {
37     ab1=MatrixMultiply(a1, b);
38     ab2=MatrixMultiply(a2, b);
39   }
40 }
```

Figure 1: MatrixMultiply PetaBricks source code

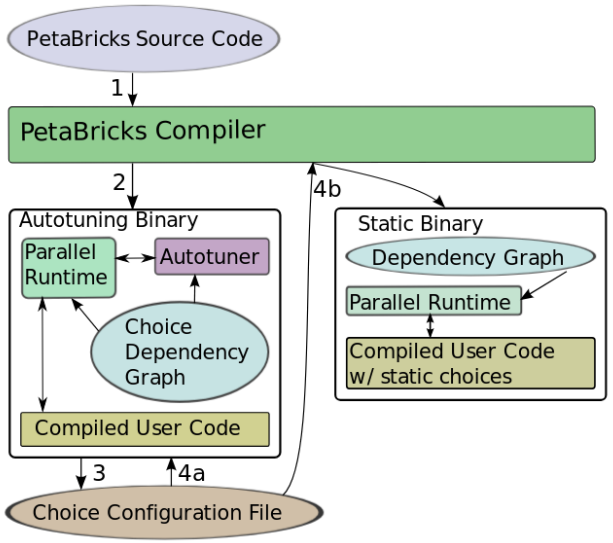

Figure 2: Interactions between compiler and output binaries.

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.
- Bottom-up finds the optimal solution for the smallest subproblem.

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.
- Bottom-up finds the optimal solution for the smallest subproblem.
- Tracks an optimal family.

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.
- Bottom-up finds the optimal solution for the smallest subproblem.
- Tracks an optimal family.
- Large problems solutions built from smaller problem solutions.

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.
- Bottom-up finds the optimal solution for the smallest subproblem.
- Tracks an optimal family.
- Large problems solutions built from smaller problem solutions.
- Evolutionary strategy: candidate -> measure runtime -> random mutation (stochastic search) -> select new 'parents'.

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.
- Bottom-up finds the optimal solution for the smallest subproblem.
- Tracks an optimal family.
- Large problems solutions built from smaller problem solutions.
- Evolutionary strategy: candidate -> measure runtime -> random mutation (stochastic search) -> select new 'parents'.
- Tracks performance vs accuracy (error and convergence rate).

UNIVERSITY OF CAMBRIDGE

How does this relate to search space optimisation?

- The autotuner is a bottom-up genetic algo which uses dynamic programming to prune the search space.
- Bottom-up finds the optimal solution for the smallest subproblem.
- Tracks an optimal family.
- Large problems solutions built from smaller problem solutions.
- Evolutionary strategy: candidate -> measure runtime -> random mutation (stochastic search) -> select new 'parents'.
- Tracks performance vs accuracy (error and convergence rate).
- **Concretely**, insertion sort is introduced at $N = 68$ instead of $N = 15$. Merge sort with 4 splits instead of 2.

UNIVERSITY OF CAMBRIDGE

Results

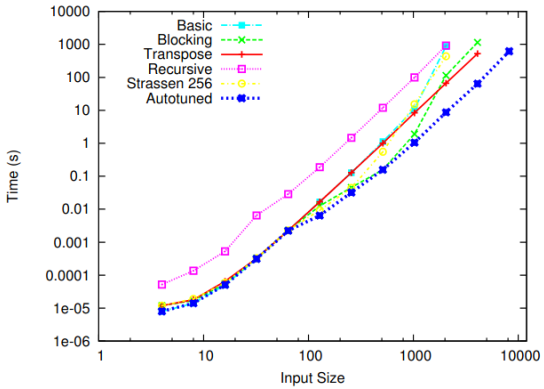

Figure 3: Matrix Multiply on 8 cores.

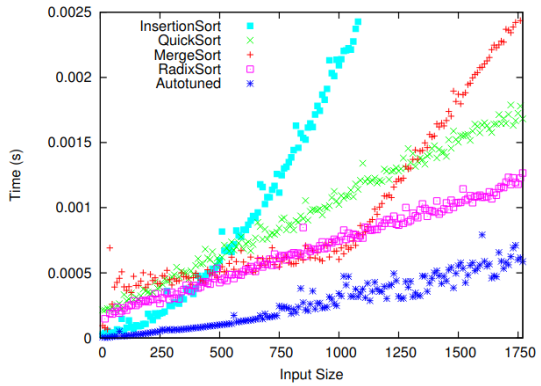

Figure 4: Sort on 8 cores.

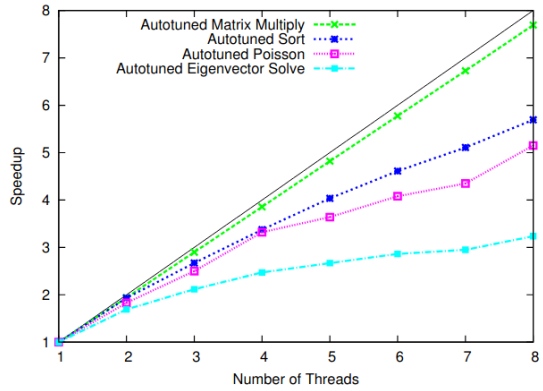

Figure 5: Parallel scalability on x86.

UNIVERSITY OF CAMBRIDGE

My Thoughts

- Maybe it can be a C++ library now (lower adoption barrier).

UNIVERSITY OF CAMBRIDGE

My Thoughts

- Maybe it can be a C++ library now (lower adoption barrier).
- Compilation takes hours on a 'good?' machine. Empirically finding parameters is time consuming.

UNIVERSITY OF CAMBRIDGE

My Thoughts

- Maybe it can be a C++ library now (lower adoption barrier).
- Compilation takes hours on a 'good?' machine. Empirically finding parameters is time consuming.
- Let's see how the autotuner really performs on data structures that aren't matrices.

UNIVERSITY OF CAMBRIDGE

My Thoughts

- Maybe it can be a C++ library now (lower adoption barrier).
- Compilation takes hours on a 'good?' machine. Empirically finding parameters is time consuming.
- Let's see how the autotuner really performs on data structures that aren't matrices.
- Should we optimise for energy consumption? final binary size? robustness (average runtime on different input data)? [2]

UNIVERSITY OF CAMBRIDGE

My Thoughts

- Maybe it can be a C++ library now (lower adoption barrier).
- Compilation takes hours on a 'good?' machine. Empirically finding parameters is time consuming.
- Let's see how the autotuner really performs on data structures that aren't matrices.
- Should we optimise for energy consumption? final binary size? robustness (average runtime on different input data)? [2]
- The core assumption is the 'Optimal Substructure Principle' [3].

UNIVERSITY OF CAMBRIDGE

My Thoughts

- Maybe it can be a C++ library now (lower adoption barrier).
- Compilation takes hours on a 'good?' machine. Empirically finding parameters is time consuming.
- Let's see how the autotuner really performs on data structures that aren't matrices.
- Should we optimise for energy consumption? final binary size? robustness (average runtime on different input data)? [2]
- The core assumption is the 'Optimal Substructure Principle' [3].
- My personal bias is against general frameworks (80/20 rule).

UNIVERSITY OF CAMBRIDGE

References I

J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe.
Petabricks: a language and compiler for algorithmic choice.
In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, page 38–49, New York, NY, USA, 2009. Association for Computing Machinery.

J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe.
Opentuner: an extensible framework for program autotuning.
In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, page 303–316, New York, NY, USA, 2014. Association for Computing Machinery.

UNIVERSITY OF CAMBRIDGE

References II


T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.
Introduction to Algorithms, Third Edition.
The MIT Press, 3rd edition, 2009.


M. Frigo and S. G. Johnson.
The fastest fourier transform in the west.
Technical report, USA, 1997.


R. C. Whaley and J. J. Dongarra.
Automatically tuned linear algebra software.
In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, page 1–27, USA, 1998. IEEE Computer Society.

UNIVERSITY OF CAMBRIDGE

Q & A

UNIVERSITY OF CAMBRIDGE