Verified Validation for Polyhedral Scheduling

Xuyang Li, Hongjin Liang, Xinyu Feng
Nanjing University
Background

• Human needs highly optimized compilation techniques for modern software.

• **Nested loops** are optimization targets due to its heavy numerical computation.

• Common loop optimizations:
  • Apply combination of loop fusion, distribution, interchange, skewing, reverse, tiling, ..., to improve memory locality and/or parallelizability.
    • May further apply vectorization (SIMD) and parallelization (openmp primitives).
  • Map it automatically to domain-specific hardware like GPU.
Background

• To do such loop transformations, we need to analyze dependences between instructions of different iterations.

```python
for i in [1, N]:
    for j in [1, N]:
```

Can we parallelize it?
for i in [1, N]:
    for j in [1, N]:
for $i$ in $[1, N]$:
    for $j$ in $[1, N]$:
Example

for i in [1, N]:
    for j in [1, N]:
Example

for i in [1, N]:
    for j in [1, N]:
Example

Can we parallelize it?
Respecting dependences?

for i in [1, N]:
    for j in [1, N]:
Example

Can we parallelize it?

Respecting dependences?
- Not along j. There is inner dependence.
- What about other axis?

```
for i in [1, N]:
    for j in [1, N]:
```
Can we parallelize it?

Respecting dependences?
- Not along j. There is inner dependence.
- What about other axis?
  - Iteration with same i + j is parallelizable

for i in [1, N]:
    for j in [1, N]:
Example

Now we may find new execution order. (Without breaking dependences)

for i in [1, N]:
    for j in [1, N]:
Now we generate a new nested loop respecting new execution order.

for \( j' \) in \([1, 2N-1]\):
  for \( i' \) in \([\max(1, j'-N+1), \min(N-1, j'-1)]\):
    S: \( A[i'][(j'-i')] = A[i'+1][(j'-i')] + A[i'-1][(j'-i')-1] \)
Example

for $i$ in $[1, N]$:
  for $j$ in $[1, N]$:

for $j'$ in $[1, 2N-1]$:
  for $i'$ in $[\max(1, j'-N+1), \min(N-1, j'-1)]$:
      $S$: $A[i'][j'-(i-1)] = A[i'+1][j'-(i-1)] + A[i'-1][j'-(i-1)-1]$

loop skewing + loop interchange
for $i$ in $[1, N]$:
  for $j$ in $[1, N]$:

for $j'$ in $[1, 2*N-1]$:
  for $i'$ in $[\max(1, j'-N+1), \min(N-1, j'-1)]$:
    $S$: $A[i'][j'-i'] = A[i'+1][j'-i'] + A[i'-1][j'-i'-1]$
Polyhedral Model

for $i$ in $[1, N]$:
    for $j$ in $[1, N]$:

for $j'$ in $[1, 2N-1]$:
    for $i'$ in $[\text{max}(1, j'-N+1), \text{min}(N-1, j'-1)]$:
        $S$: $A[i'][j'-i'] = A[i'+1][j'-i'] + A[i'-1][j'-i'-1]$

loop skewing + loop interchange
Polyhedral Compilation

for $i$ in $[1, N]$:
  for $j$ in $[1, N]$:

for $j'$ in $[1, 2*N-1]$:
  for $i'$ in $[\max(1, j'-N+1), \min(N-1, j'-1)]$:
    $S$: $A[i'][(j'-i')] = A[i'+1][(j'-i')] + A[i'-1][(j'-i')-1]$
Polyhedral Compilation
Polyhedral Compilation

Our focus!
Scheduling can be done manually or by automatic algorithms, like Pluto\textsuperscript{1}. We want to ensure its correctness.

\textsuperscript{1} https://pluto-compiler.sourceforge.net/
Compilation correctness

• For compiler $\text{Comp}$, programs $\mathcal{P}_s$ and $\mathcal{P}_t$ that $\text{Comp}(\mathcal{P}_s) = \text{Some } \mathcal{P}_t$.

• If $\mathcal{P}_t \textbf{ refines } \mathcal{P}_s$ (written as $\mathcal{P}_t \sqsubseteq \mathcal{P}_s$), we say this compilation is correct.
  • It says, from the same beginning state, whenever $\mathcal{P}_t$ terminates at some state, then $\mathcal{P}_s$ is able to stop at the same final state.
Compilation correctness

• For compiler \( \text{Comp} \), programs \( P_s \) and \( P_t \) that \( \text{Comp}(P_s) = \text{Some } P_t \).

• If \( P_t \) refines \( P_s \) (written as \( P_t \subseteq P_s \)), we say this compilation is correct.
  • It says, from the same beginning state, whenever \( P_t \) terminates at some state, then \( P_s \) is able to stop at the same final state.

• Two ways to guarantee correct compilation:
  • Compiler proof: reasoning on \( \text{Comp} \)'s concrete definition to prove
    \[
    \forall P_s, P_t. \text{Comp}(P_s) = \text{Some } P_t \implies P_t \subseteq P_s.
    \]
  • Verified validation: define a separate validator \( \text{Validate} \) and prove
    \[
    \forall P_s, P_t. \text{Validate}(P_s, P_t) = \text{true} \implies P_t \subseteq P_s.
    \]
  • And run \( \text{Validate} \) after each run of \( \text{Comp} \).
Compilation correctness

Why not directly verify the scheduling algorithm?

- On the one hand, it contains complex heuristic with heavy mathematics. Hard/Impractical to verify.
- On the other hand, it has simple validation algorithm due to its simple correctness criterion: not breaking dependence. (Also called Bernstein’s Condition\(^2\).)

• Two ways to guarantee correct compilation:
  • Compiler proof: reasoning on \(\text{Comp}\)'s concrete definition to prove
    \[
    \forall P_s, P_t. \:\text{Comp}(P_s) = \text{Some} \: P_t \implies P_t \subseteq P_s.
    \]
  • Verified validation: define a separate validator \(\text{Validate}\) and prove
    \[
    \forall P_s, P_t. \: \text{Validate}(P_s, P_t) = \text{true} \implies P_t \subseteq P_s.
    \]
  • And run \(\text{Validate}\) after each run of \(\text{Comp}\).

\(^2\) https://link.springer.com/referenceworkentry/10.1007/978-3-8766-4_521
Validating Polyhedral Scheduling

Output: true or false (unknown).
Our work: 
Verified Validation for Polyhedral Scheduling

• We implement and verify a general validation function \textit{Validate} for polyhedral scheduling in Coq proof assistant from scratch. It is parameterized by instruction language.

• We instantiate \textit{Validate} with a variant of \textit{CompCert}'s instruction language, showing its practicality.

• We adapt \textit{Validate} so that it works with the \textit{Pluto} compiler. It successfully validates all its available test cases with reasonable overhead.
General Verified Validation function for Polyhedral Scheduling

• We define a validation function $\text{Validate}$ that checks Bernstein's condition within polyhedral model, and mechanized its correctness with the Coq proof assistant.

• It is parametrized by instruction language to be reusable.

• Our top lemma

Definition (correctness of the validator)

$$\text{Correct}(\text{Validate}) \triangleq \forall \mathcal{P}_s, \mathcal{P}_t. \text{Validate}(\mathcal{P}_s, \mathcal{P}_t) = \text{true}$$

$$\implies \mathcal{P}_t \subseteq \mathcal{P}_s.$$
Case study on CompCert

• What is CompCert\(^3\)?
  • a high-assurance compiler for almost all of the C language (ISO C 2011), generating efficient code for the ARM, PowerPC, RISC-V and x86 processors.
  • It is not enough optimized\(^4\) than production compiler like clang and gcc. So aggressive optimizers like polyhedral-based ones do help!

• We successfully instantiate Validate with CompCert’s semantics model, showing the possibility towards a fully verified polyhedral extension to CompCert.

\(^3\)https://compcert.org/

\(^4\)https://doi.org/10.1145/3622799
Case study on Pluto compiler

• Loop optimizers like polyhedral-based ones are error prone indeed! So formal methods do help.

• We evaluate on Pluto, one of the famous polyhedral compiler.
  • Pluto: ACM SIGPLAN PLDI Most Influential Paper award in 2018

Figure from https://www.csa.iisc.ac.in/~udayb/publications/uday-thesis.pdf, Page 98

\(^5\) https://doi.org/10.1145/3591295
Case study on Pluto compiler

• Loop optimizers like polyhedral-based ones are error prone\(^5\) indeed! **So formal methods do help.**

• We evaluate on Pluto, one of the famous polyhedral compiler.
  • Pluto: ACM SIGPLAN PLDI Most Influential Paper award in 2018

\(^5\)https://doi.org/10.1145/3591295
Case study on Pluto compiler

• Loop optimizers like polyhedral-based ones are error prone indeed! So formal methods do help.

• We evaluate on Pluto, one of the famous polyhedral compiler.
  • Pluto: ACM SIGPLAN PLDI Most Influential Paper award in 2018

• Work and work well with Pluto compiler: successfully validate its all 62 available test cases.
  • Not just a research prototype: work as an executable on practical polyhedral compiler.
  • Soundness v.s. completeness:
    • Formal verification proves the algorithm soundness: guarantee refinement whenever it outputs true.
    • The evaluation shows validation’s algorithm completeness: if refinement establishes on inputs, the algorithm tries its best to output true rather than unknown!
  • Also, the algorithm has reasonable overhead.

https://doi.org/10.1145/3591295
Case study on Pluto compiler

- Loop optimizers like polyhedral ones are error prone indeed!
- So formal methods do help.
- We evaluate on Pluto, one of the famous polyhedral compiler.
- Pluto: ACM SIGPLAN PLDI Most Influential Paper award in 2018
- Work and work well with Pluto compiler: successfully validate its all 62 available test cases.
- Not just a research prototype: work as an executable on practical polyhedral compiler.
- Soundness vs. completeness:
  - Formal verification proves the algorithm soundness: guarantee refinement whenever it outputs true.
  - The evaluation shows validation's algorithm completeness: if refinement establishes on inputs, the algorithm tries its best to output true rather than unknown!
- Also, the algorithm has reasonable overhead.

<table>
<thead>
<tr>
<th>Test</th>
<th>Time of Pluto (ms)</th>
<th>Time of Validation (ms,ms)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>covcol</td>
<td>3.5</td>
<td>434.6, 320.7</td>
<td>EQ</td>
</tr>
<tr>
<td>dsyr2k</td>
<td>2.6</td>
<td>106.0, 83.4</td>
<td>EQ</td>
</tr>
<tr>
<td>ftdt-2d</td>
<td>46.4</td>
<td>1615.5, 1296.3</td>
<td>EQ</td>
</tr>
<tr>
<td>gemver</td>
<td>7.0</td>
<td>247.9, 240.4</td>
<td>EQ</td>
</tr>
<tr>
<td>lu</td>
<td>6.1</td>
<td>410.6, 331.2</td>
<td>EQ</td>
</tr>
<tr>
<td>mvt</td>
<td>2.2</td>
<td>70.2, 56.3</td>
<td>EQ</td>
</tr>
<tr>
<td>ssymm</td>
<td>40.7</td>
<td>726.0, 551.2</td>
<td>EQ</td>
</tr>
<tr>
<td>tce</td>
<td>568.6</td>
<td>4442.0, 4422.5</td>
<td>EQ</td>
</tr>
<tr>
<td>adi</td>
<td>77.5</td>
<td>2531.7, 2377.8</td>
<td>EQ</td>
</tr>
<tr>
<td>corcol</td>
<td>5.5</td>
<td>442.5, 362.1</td>
<td>EQ</td>
</tr>
<tr>
<td>dct</td>
<td>21.8</td>
<td>879.4, 739.4</td>
<td>EQ</td>
</tr>
<tr>
<td>dsyrk</td>
<td>1.8</td>
<td>96.8, 78.9</td>
<td>EQ</td>
</tr>
<tr>
<td>floyd</td>
<td>12.1</td>
<td>502.6, 421.7</td>
<td>EQ</td>
</tr>
<tr>
<td>jacobi-1d-imper</td>
<td>3.8</td>
<td>184.0, 167.8</td>
<td>EQ</td>
</tr>
<tr>
<td>matmul-init</td>
<td>2.9</td>
<td>257.8, 192.4</td>
<td>EQ</td>
</tr>
<tr>
<td>pca</td>
<td>202.5</td>
<td>2923.6, 2679.5</td>
<td>EQ</td>
</tr>
<tr>
<td>strmm</td>
<td>1.9</td>
<td>141.4, 110.8</td>
<td>EQ</td>
</tr>
<tr>
<td>tmv</td>
<td>1.6</td>
<td>109.7, 89.6</td>
<td>EQ</td>
</tr>
<tr>
<td>advect3d</td>
<td>1023.1</td>
<td>579.1, 498.1</td>
<td>EQ</td>
</tr>
<tr>
<td>corcol3</td>
<td>13.6</td>
<td>851.3, 733.4</td>
<td>EQ</td>
</tr>
<tr>
<td>doitgen</td>
<td>10.4</td>
<td>1069.2, 837.4</td>
<td>EQ</td>
</tr>
<tr>
<td>ftdt-1d</td>
<td>6.0</td>
<td>268.7, 229.9</td>
<td>EQ</td>
</tr>
<tr>
<td>jacobi-2d-imper</td>
<td>17.7</td>
<td>619.5, 543.5</td>
<td>EQ</td>
</tr>
<tr>
<td>matmul</td>
<td>3.2</td>
<td>157.1, 125.5</td>
<td>EQ</td>
</tr>
<tr>
<td>seidel</td>
<td>24.5</td>
<td>818.1, 725.5</td>
<td>EQ</td>
</tr>
<tr>
<td>strsm</td>
<td>6.4</td>
<td>209.3, 161.2</td>
<td>EQ</td>
</tr>
<tr>
<td>trisolv</td>
<td>5.1</td>
<td>338.9, 248.8</td>
<td>EQ</td>
</tr>
<tr>
<td>1dloop-invar</td>
<td>0.3</td>
<td>6.7, 6.0</td>
<td>EQ</td>
</tr>
<tr>
<td>costfunc</td>
<td>0.8</td>
<td>47.4, 35.0</td>
<td>EQ</td>
</tr>
<tr>
<td>fusion1</td>
<td>0.9</td>
<td>15.3, 13.9</td>
<td>EQ</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1: Evaluation results on Pluto’s test suits
Coq development

• Around 18000 lines of Coq, 1000 lines of OCaml. Open source at https://github.com/verif-scop/PolCert.

• Our work bases on Verified Polyhedron Library\textsuperscript{6} and is syntactically compatible to PolyGen\textsuperscript{7} (Verified Polyhedral Code Generation) in POPL'21.

\textsuperscript{6}https://ieeexplore.ieee.org/document/8750763
\textsuperscript{7}https://dl.acm.org/doi/10.1145/3434321
Future work

• Complete verified polyhedral compilation.
  • Verified extractor.
  • Engineering in CompCert’s driver & frontend.
  • Apply optimistic approach\(^8\) to deal with polyhedral model's heavy assumptions, like integer overflow\(^9\).

• Support validation for other polyhedral transformations, like index set split (as a pre-phrase), tiling (as a post-phrase), layout transformation (as an orthogonal phrase).

• Support vectorization, parallelization, GPU compilation ...

\(^8\) https://dl.acm.org/doi/10.5555/3049832.3049864
\(^9\) https://inria.hal.science/hal-00655485
Q&A
Our work:
Verified Validation for Polyhedral Scheduling

• We implement and verify a general validation function *Validate* for polyhedral scheduling in Coq proof assistant from scratch. It is parameterized by instruction language.

• We instantiate *Validate* with a variant of *CompCert*'s instruction language, showing its practicality.

• We adapt *Validate* so that it works with the *Pluto* compiler. It successfully validates all its available test cases with reasonable overhead.
Original code $\pi_{cov}$ for covariance matrix calculation, 1.84s

```c
for (j1 = 1; j1 <= M; j1++) {
    for (j2 = j1; j2 <= M; j2++) {
        for (i = 1; i <= N; i++) {
            I_0: symmat[j1][j2] += data[i][j1] * data[i][j2];
        }
    }
    I_1: symmat[j2][j1] = symmat[j1][j2];
}
```

Optimized code $\pi'_{cov}$ for covariance matrix calculation, 0.43s, with loop distribution and loop interchange

```c
for (i = 1; i <= N; i++) {
    for (j1 = 1; j1 <= M; j1++) {
        for (j2 = j1; j2 <= M; j2++) {
            symmat[j1][j2] += data[i][j1] * data[i][j2];
        }
    }
}
for (j1 = 1; j1 <= M; j1++) {
    for (j2 = j1; j2 <= M; j2++) {
        symmat[j2][j1] = symmat[j1][j2];
    }
}
```

Polyhedral compilation does high-level structural transformations and only impose a few properties of the underlying instruction language (called II). The validation function given in this work is parameterized by II. II allows user define the syntax, types, state, semantics of the language, how it initializes, and its the non-alias proposition. It demands user to provide a verified Checker function to validate the consistency between the read and write access function and an instruction’s semantics, and prove that any two instances that satisfy Bernstein’s conditions are permutable.
Module Type $\triangleq$

$T, I, S : Type$

$\models : I \rightarrow List(\mathbb{Z}) \rightarrow Memory Cells$

$\rightarrow Memory Cells \rightarrow S \rightarrow S \rightarrow Prop$

Compat : List(Identifier) $\rightarrow S \rightarrow Prop$

Consistent : List(Identifier $\times$ T) $\rightarrow List(\mathbb{Z}) \rightarrow S \rightarrow Prop$

NonAlias : S $\rightarrow Prop$

NonAliasPsrv : $\forall I, \sigma, \sigma'. \text{NonAlias}(\sigma) \land p \models I, \sigma \overset{\sigma'}{\rightarrow} \sigma' \implies \text{NonAlias}(\sigma').$

Checker : I $\rightarrow Access Functions$

$\rightarrow Access Functions \rightarrow Bool$

Correct (Checker) : $\forall I, \mathcal{W}, \mathcal{R}. \text{Checker}(I, \mathcal{W}, \mathcal{R}) = true$

$\implies (\forall \sigma, \sigma', p, \Delta_r, \Delta_w. p \models I, \sigma \overset{\Delta_r, \Delta_w}{\rightarrow} \sigma' \implies \Delta_r \subseteq \mathcal{R}(p) \land \Delta_w \subseteq \mathcal{W}(p)).$

BCPermut : $\forall I_1, I_2, p_1, p_2, \sigma, \sigma', \sigma'', \Delta_r, \Delta_w, \Delta'_r, \Delta'_w.$

$(p_1 \models I_1, \sigma \overset{\Delta_r, \Delta_w}{\rightarrow} \sigma' \land p_2 \models I_2, \sigma' \overset{\Delta'_r, \Delta'_w}{\rightarrow} \sigma'')$

$\land \Delta_r \cap \Delta'_w = \emptyset \land \Delta_w \cap \Delta'_r = \emptyset \land \Delta_w \cap \Delta'_w = \emptyset)$

$\implies \exists \sigma^*. p_2 \models I_2, \sigma \overset{\Delta'_r, \Delta'_w}{\rightarrow} \sigma^* \land p_1 \models I_1, \sigma^* \overset{\Delta_r, \Delta_w}{\rightarrow} \sigma''.$

Figure 1: Definition of Instruction Language Module II

We assume Pluto’s instruction language satisfy this abstraction.
Case study: Towards verified polyhedral compilation for CompCert

We instantiate the validation function with CompCert C’s type, state and subset of its instruction language, and implement Checker with symbolic execution. All verified. Only differences are, affine expression is evaluated in $\mathbb{Z}$ (no overflow), and multi-dimensional array access is sugarized.

$$
\begin{align*}
\text{(Base Type)} & \quad \tau_\bot := \text{int32s} \\
\text{(Type)} & \quad \tau \in \text{Base Type} \times \text{List}(\mathbb{Z}) \\
\text{(Value)} & \quad v := \text{I32}(n) | \ldots \\
\text{(Iterator)} & \quad i \in \mathbb{N} \\
\text{(Unop)} & \quad \text{op}_1 := - | \ldots \\
\text{(Binop)} & \quad \text{op}_2 := + | \ast | \ldots \\
\text{(May Affine Expression)} & \quad \varepsilon := z | i | \text{op}_1 \varepsilon | \varepsilon_1 \text{op}_2 \varepsilon_2 \\
\text{(Access Expression)} & \quad \epsilon \in \text{Identifier} \times \text{Base Type} \times \text{List}(\text{May Affine Expression}) \\
\text{(Expression)} & \quad e := v | i | \epsilon | \text{op}_1 e | e_1 \text{op}_2 e_2 \\
\text{(Base Instruction)} & \quad I := \text{skip} | \epsilon := e
\end{align*}
$$
Polyhedral Compilation

• Extraction -> Scheduling -> Codegen

loop reverse + loop fusion

for i in [0, 3]:
S:   B[i] = A[i] + 1
for i in [0, 3]:
T:   C[i] = B[2-i] * 3

~>
for i in [0, 3]:
S:   B[i] = A[i] + 1
T:   C[2-i] = B[i] * 3
Polyhedral Model

• An alternative integer-programming-based representation for nested loop. A **polyhedral program** is a multiset of **polyhedral instructions**, which consists of at least three parts: the **instruction**, the **domain**, and the **schedule**.

• “Extraction”

```
for i in [0, 3]:  ~> 
S:   B[i] = A[i] + 1  { (S, 0<=i<3, [i]~->[0,i]),
for i in [0, 3]:  ~> 
T:   C[i] = B[2-i] * 3
```

Execution order: S[0];S[1];S[2];T[0];T[1];T[2]
Polyhedral Model

Our work’s focus!

• “Scheduling”: do reordering transformation on polyhedral model, guided by dependence analysis.

• Well-designed automatic algorithms, like Pluto\(^2\), serve for this purpose.

• Manual scheduling is also possible.

• Recall the correctness criterion: Bernstein’s condition.

\{(S, 0\leq i<3, \[i\] \sim \>[0,i]), \sim \> \{ (S, 0\leq i<3, \[i\] \sim \>\ldots), \}

(\[T, 0\leq i<3, \[i\] \sim \>[1,i])\} \quad (T, 0\leq i<3, \[i\] \sim \>\ldots)\}

Old Execution order: \(S[0]; S[1]; S[2]; T[0]; T[1]; T[2]\)

\(^2\) https://pluto-compiler.sourceforge.net/
Polyhedral Model

- “Scheduling”: do reordering transformation on polyhedral model, guided by dependence analysis.
- Well-designed automatic algorithms, like Pluto\(^2\), serve for this purpose.
- Manual scheduling is also possible.

Average distance of dependence’s satisfaction seems too long. Cache locality.

Old Execution order: \( S[0]; S[1]; S[2]; T[0]; T[1]; T[2] \)

\(^2\)https://pluto-compiler.sourceforge.net/
Polyhedral Model

Our work’s focus!

- “Scheduling”: do reordering transformation on polyhedral model, guided by dependence analysis.
- Well-designed automatic algorithms, like Pluto\(^2\), serve for this purpose.
- Manual scheduling is also possible.

Average distance of dependence’s satisfaction seems to long.

Old Execution order: \(S[0]; S[1]; S[2]; T[0]; T[1]; T[2]\)

New execution order: \(S[0]; T[2]; S[1]; T[1]; S[2]; T[0]\)

Try to minimize the distance without breaking dependences!

\(^2\)https://pluto-compiler.sourceforge.net/
Polyhedral Model

Our work’s focus!

• “Scheduling”: do reordering transformation on polyhedral model, guided by dependence analysis.

• Well-designed automatic algorithms, like Pluto\textsuperscript{2}, serve for this purpose.

• Manual scheduling is also possible.

• Recall the correctness criterion: Bernstein’s condition.

\[
\{(S, 0\leq i<3, [i] \rightarrow [0,i]), ~> ~} \{ (S, 0\leq i<3, [i] \rightarrow [i,0]), \\
(T, 0\leq i<3, [i] \rightarrow [1,i]))\} \quad (T, 0\leq i<3, [i] \rightarrow [2-i,1])\}

Old Execution order: \( S[0]; S[1]; S[2]; T[0]; T[1]; T[2] \)

New execution order: \( S[0]; T[2]; S[1]; T[1]; S[2]; T[0] \)

\textsuperscript{2} https://pluto-compiler.sourceforge.net/
Polyhedral Model

• “Codegen”: recover the imperative control structure from the optimized polyhedral model.

\{(S, \ 0 \leq i < 3, \ [i] \rightarrow [i, 0]),
(T, \ 0 \leq i < 3, \ [i] \rightarrow [2-i, 1])\} \rightarrow 

for i in [0, 3]:
S: \ B[i] = A[i] + 1
T: \ C[2-i] = B[i] * 3