Collaborative Parallelization Framework

Sotiris Apostolakis, Greg Chan, Ziyang Xu, Benjamin Huang, and David I. August

Liberty Research Group
Princeton University
Automatic parallelization is great ...

... when it works

How to compose?

**E<sub>i</sub>:** Enabling Transformations (e.g., Memory Speculation)

**P<sub>i</sub>:** Parallelization Techniques (e.g., DOALL, PS-DSWP)
Every transformation is protected by two guards…

[Drawings courtesy of Nick P. Johnson]
Either may reject a program

[Drawings courtesy of Nick P. Johnson]
We can gather wisdom from them

Is it applicable / profitable?  Why is it not applicable / profitable?

[Drawings courtesy of Nick P. Johnson]
Critic: Answers why a parallelization technique is not applicable/profitable?
Remediator:
- Uses applicability guard of enabling transformations.
- Ignores original profitability guard; the transformation is useful if a criticism is satisfied
- Do not apply the transformation but express its effect
Collaborative Parallelization Framework

**E_i**: Enabling Transformations

**P_i**: Parallelization Techniques

**R_i**: Remediator

**C_i**: Critic

![Diagram of Collaborative Parallelization Framework](image-url)

**Planning Phase**

**Transformation Phase**
Collaborative Parallelization Framework

Parallelizing Compiler 1

$E_1 \rightarrow E_2 \rightarrow P_1$

Parallelizing Compiler 2

$E_3 \rightarrow E_1 \rightarrow P_2$

Orchestrator

$R_1 \rightarrow R_2 \rightarrow R_3$

Planning Phase

Transformation Phase

Reproduce existing compilers

$E_1 \rightarrow E_2 \rightarrow P_1$

$E_i$: Enabling Transformations

$P_i$: Parallelization Techniques

$R_i$: Remediator

$C_i$: Critic
Collaborative Parallelization Framework

Parallelizing Compiler 1

\[ E_1 \rightarrow E_2 \rightarrow P_1 \]

Parallelizing Compiler 2

\[ E_3 \rightarrow E_1 \rightarrow P_2 \]

? \[ Orchestrator \]

\[ R_1 \rightarrow R_2 \rightarrow R_3 \]

\[ C_1 \rightarrow C_2 \]

Planning Phase

\[ Transformation Phase \]

Create new hybrid versions

\[ E_1 \rightarrow E_3 \rightarrow E_2 \rightarrow P_1 \]

\[ E_i: Enabling Transformations \]

\[ P_i: Parallelization Techniques \]

\[ R_i: Remediator \]

\[ C_i: Critic \]
Programmer as a Remediator

**Criticisms**

- **Applicability**: Checks if criticisms can be translated to high-level yes/no questions
- **Profitability**: High-probability assumptions

Assume expected answer, ask programmer only when remedy part of final parallelization plan

**Programmer Remediator**:
- **Applicability**: Checks if criticisms can be translated to high-level yes/no questions
- **Profitability**: High-probability assumptions
Conclusion

• Combine compiler advancements on automatic parallelization into an unified compiler framework
  • Better automated and robust parallelization decision process
    • Transformations communicate through criticisms and remedies
    • New supervisory compiler component, called The Orchestrator

• Modularity
  • Easy to add new transformations to the system
  • Every transformation developed independently

• Minimize programmer involvement
  • Seek help from the programmer only when necessary
Thank you

Questions?