■ CellVM

- ヘテロジニアスアーキテクチャの上に均質な VM を作る
- Java Thread を各コアに分配する

- 現在の実装では，8 スレッドを 8 コのコアに分配する

- SPE では CoreVM が走る
  - PPE からの intervention と assistance なしに Java の bytecode を実行
  - ローカルでは実行できないときだけ ShellVM(PPE) に助けを依頼 (=> cooperative interpretation; co-interpretation と呼ぶ)

- new objects(array 含む) を作る opcode は，ShellVM で実行される
  - すべての Java heap はメインメモリにあるから
- native method 実行も ShellVM で扱われる
  - 普通，そんなにない
  - Java heap を操作することになる (array copying とか)

- JavmVM の拡張
  - direct threaded interpreter
    - 各バイトコード命令は対応するコードブロックのアドレス
    - machine-level 分岐の数 /opcode が削減される
  - opcode rewriting: to store resolved information in the operand of an instruction(Sec 3.2)

- SPE の L.S. を効率良く使うことが目標
  - DMA 転送は数百サイクルのレイテンシが必要

- 各 CoreVM は，現在の状態を表わす内部レジスタを管理する．
  - ローカル変数へのポインタ，stack top pointer，現在のフレームへの参照
- method 実行毎に Java frame 作る
  - method code はメインメモリ

■ Tech talk: Gauche Scheme

http://jp.youtube.com/watch?v=WEBOdWyGE3E とか見る．

直前の式の結果をスタックに積まずにレジスタに入れるっていうのは，
scheme だから，大きな効果があるのかな？

■ Escape analysis for object-oriented languages: application to Java

Escape analysis [27, 14, 5] is a static analysis that determines whether the lifetime of data exceeds its static scope.The main originality of our escape analysis is that it determines precisely the effect of assignments, which is necessary to apply it to object oriented languages with promising results, whereas previous work [27, 14, 5] applied it to functional languages and were very imprecise on assignments. Our implementation analyses the full Java(TM) Language.We have applied our analysis to stack allocation and synchronization elimination. We manage to stack allocate 13% to 95% of data, eliminate more than 20% of synchronizations on most programs (94% and 99% on two examples) and get up to 44% speedup (21% on average). Our detailed experimental study on large programs shows that the improvement comes from the decrease of the garbage collection and allocation times than from improvements on data locality [7], contrary to what happened for ML [5].

```
@article{320387,
author = {Bruno Blanchet},
title = {Escape analysis for object-oriented languages: application to Java},
journal = {SIGPLAN Not.},
volume = {34},
number = {10},
year = {1999},
issn = {0362-1340},
pages = {20--34},
doi = {http://doi.acm.org/10.1145/320385.320387},
publisher = {ACM},
address = {New York, NY, USA},
}
```

■ Escape analysis for Java

This paper presents a simple and efficient data flow algorithm for escape analysis of objects in Java programs to determine (i) if an object can be allocated on the stack; (ii) if an object is accessed only by a single thread during its lifetime, so that synchronization operations on that object can be removed. We introduce a new program abstraction for escape analysis, the connection graph, that is used to establish reachability relationships between objects and object references. We show that the connection graph can be summarized for each method such that the same summary information may be used effectively in different calling contexts. We present an interprocedural algorithm that uses the above property to efficiently compute the connection graph and identify the non-escaping objects for methods and threads. The experimental results, from a prototype implementation of our framework in the IBM High Performance Compiler for Java, are very promising. The percentage of objects that may be allocated on the stack exceeds 70% of all dynamically created objects in three out of the ten benchmarks (with a median of 19%), 11% to 92% of all lock operations are eliminated in those ten programs (with a median of 51%), and the overall execution time reduction ranges from 2% to 23% (with a median of 7%) on a 333 MHz PowerPC workstation with 128 MB memory.

```
@article{320386,
 author = {Jong-Deok Choi and Manish Gupta and Mauricio Serrano and Vugranam C. Sreedhar and Sam
Midkiff},
 title = {Escape analysis for Java},
 journal = {SIGPLAN Not.},
 volume = {34},
```

```
number = {10},
year = {1999},
issn = {0362-1340},
pages = {1--19},
doi = {http://doi.acm.org/10.1145/320385.320386},
publisher = {ACM},
address = {New York, NY, USA},
}
```