次世代超並列計算機開発
― 連続体向け超並列計算機の開発 ―

朴 泰祐

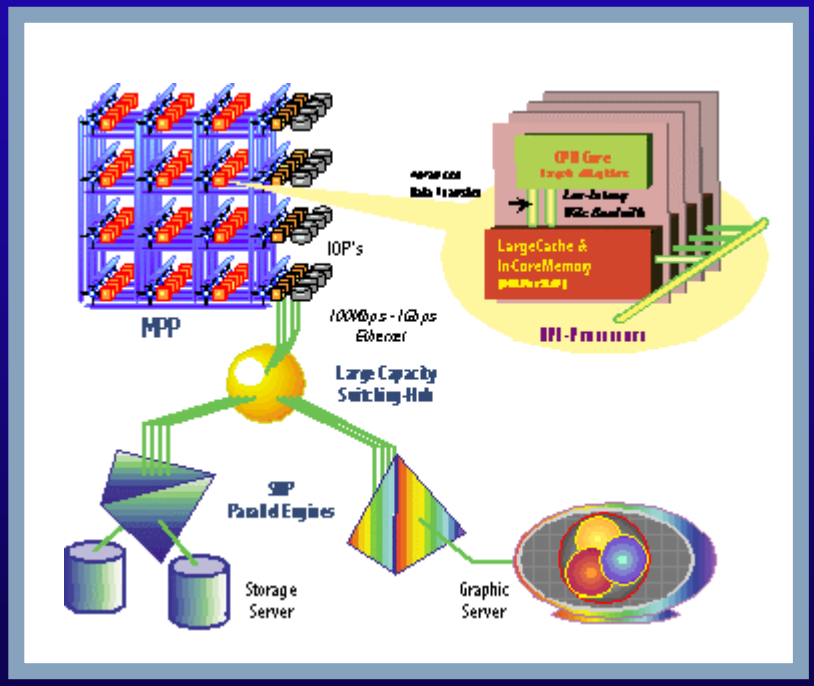
筑波大学電子・情報工学系／計算物理学研究センター

次世代超並列計算機の開発

- 連続体向け超並列計算機の開発
- 多粒子系連続体向け超並列計算機の開発

計算科学のための次世代超並列計算機

- 演算性能のさらなる追求
  - 単体プロセッサ性能の向上
  - ピーク性能ではなく実効性能を
  - ノード構成
  - 相互結合網
- 計算機の利用性・柔軟性の向上
  - 入出力の強化
  - 演算結果の前処理・後処理
  - 演算結果に対する解釈



連続体向け超並列計算機
システムの全体イメージ

研究組織

岩崎洋一　（筑波大学副学長）

計算機工学

- 坂井　修一　（東京大学工学部）
- 朴　泰祐　（筑波大学電子・情報学系）
- 山下　義行　（筑波大学電子・情報学系）
- 和田　耕一　（筑波大学電子・情報学系）
- 安永　守利　（筑波大学電子・情報学系）
- 千葉　滋　（筑波大学電子・情報学系）
- 星野　力　（筑波大学構造工学学系）
- 白川　友紀　（筑波大学構造工学学系）
- 中村　宏　（東京大学先端科学研究所）
- 渡瀬　芳行　（高エネルギー加速器研究機構）
- 中澤　喜三郎　（明星大学情報学部）
- 中田　育男　（図書館情報大学）

計算物理学

- 宇川　彰　（筑波大学物理学系）
- 金谷　和至　（筑波大学物理学系）
- 青木　慎也　（筑波大学物理学系）
- 吉江　友照　（筑波大学物理学系）
- 梅村　雅之　（筑波大学物理学系）
- 中本　泰史　（筑波大学物理学系）
- 大川　正典　（高エネルギー加速器研究機構）

並列入力・並列可視化

プロセッサアーキテクチャ

プロセッサアーキテクチャの要件

- 非常に高密度なトランジスタ実装
- チップ内周波数の増加 (v.s. チップ間周波数)
- メモリアクセス・レイテンシの (相対的な) 増加

⇓

- ☆ 最新のVLSI技術を活かした命令レベル並列性(ILP)
- ☆ 演算器を休みなく稼動させるための十分なデータ供給能力
- ☆ データ局所性の有効利用

⇒ オンチップメモリを利用した高並列プロセッサ

SIA予測

Year	1997	2001	2003	2006?	2009??
Rule (nm)	200	120	100	70	35
Wafer(mm^2)	300	385	430	520	750
Layer	6	7	7	7-8	9
Tr./ch	11M	40M	76M	200M	1.40G
Pins	1100	1800	2200	3000	5500
Chip I/Os	1450	2400	3000	4000	7300
Chip clock(Hz)	750M	1.5G	2.1G	3.5G	10G
I/O clock (Hz)	750M	1.4G	1.6G	2.0G	3.0G
Voltage (V)	1.8-2.5	1.2-1.5	1.2-1.5	0.9-1.2	0.5-0.6
Power (W)	70	110	130	160	175

オフチップ･データ転送性能を向上させるためには
バルク転送によるスループットの向上が必要不可欠

キャッシュ v.s. オンチップメモリ

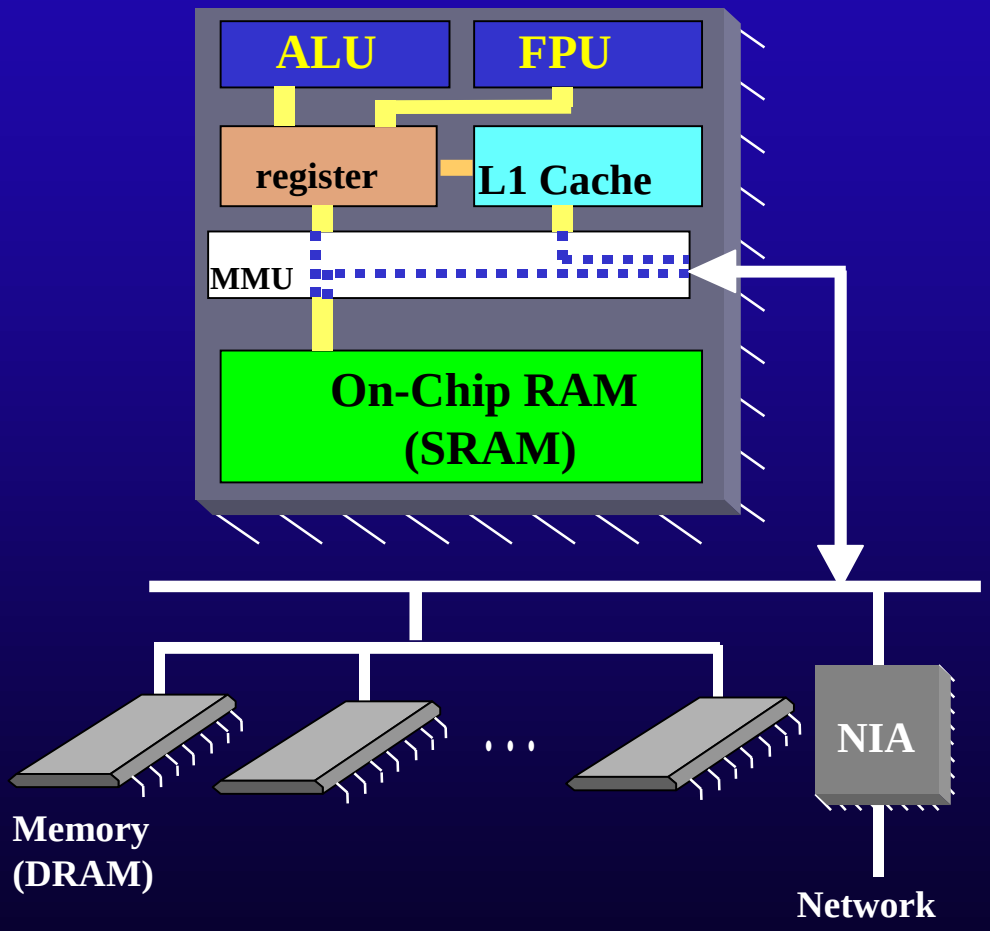
－ キャッシュ －	－ オンチップメモリ －
ハードウェア制御でありデータアロケーション、及びリプレースメントが自動的に行われる	ソフトウェア制御であり通常のメモリと同様ソフトウェアで明示的に制御

- **キャッシュではユーザが意図しないデータアロケーション、リプレースメントによる性能低下が問題になる**
  （ブロッキングなどがスムーズにいかない）

SCIMA

(Software Controlled Integrated Memory Architecture)

SCIMAアーキテクチャの概要



SCIMAアーキテクチャのモデル図

- チップ内：キャッシュ以外にオンチップメモリも搭載
  容量よりもレイテンシを重視⇒SRAMを想定
- チップ外：ＨＰＣの大規模データセットを想定しチップ外にもオフチップメモリ（ＤＲＡＭ）を配置
- 必要に応じSMP化も検討

オンチップメモリの利用

- オンチップメモリを用いることでキャッシュの問題点に対処
- アクセスが定型的なものは

  register ↔ on-chip RAM ↔ off-chip RAM
- アクセスが定型的でないものは

  register ↔ cache ↔ off-chip RAM

HPCでは定型的なデータアクセスが多く、オンチップメモリを用いることで効率的なデータ転送ができる

- データの再利用、プリフェッチが効率的になる
- バンクコンフリクトの多発を防ぐ

アドレス空間

- オンチップメモリ／オフチップメモリという2つの性質の異なるメモリを扱う必要あり

↓

- アドレス空間で両者を区別

↓

物理アドレス空間上の特定のオンチップメモリ空間を論理アドレスにマッピングするのが妥当

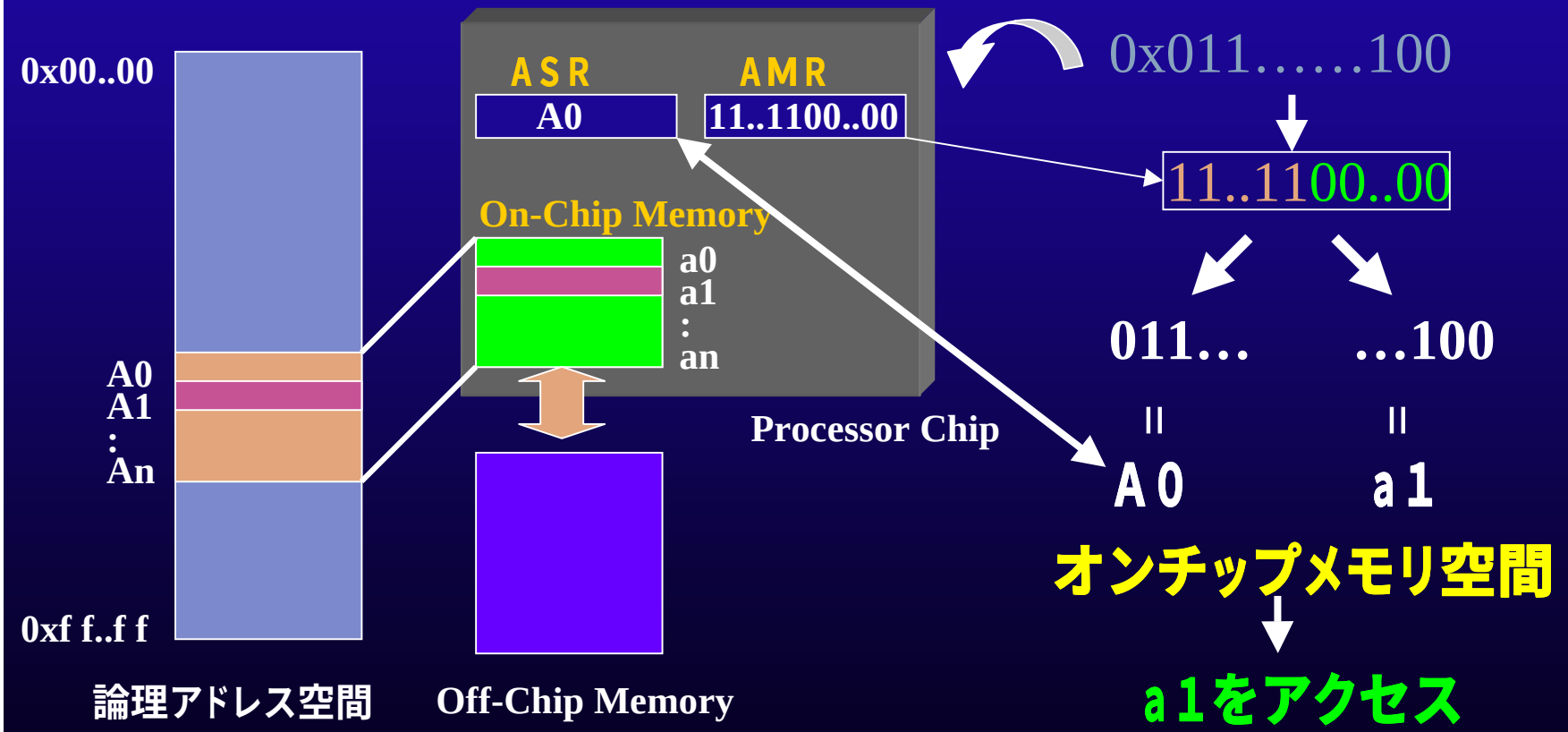
- 一般的な方法：ページテーブル（TLB）でアドレス変換時にオンチップメモリアドレスかどうかを管理する

→
- × オンチップメモリ領域は連続するブロック領域のため、TLBでの管理は効率が悪い
- × オンチップメモリ領域でTLBミス発生の可能性あり

オンチップメモリの管理

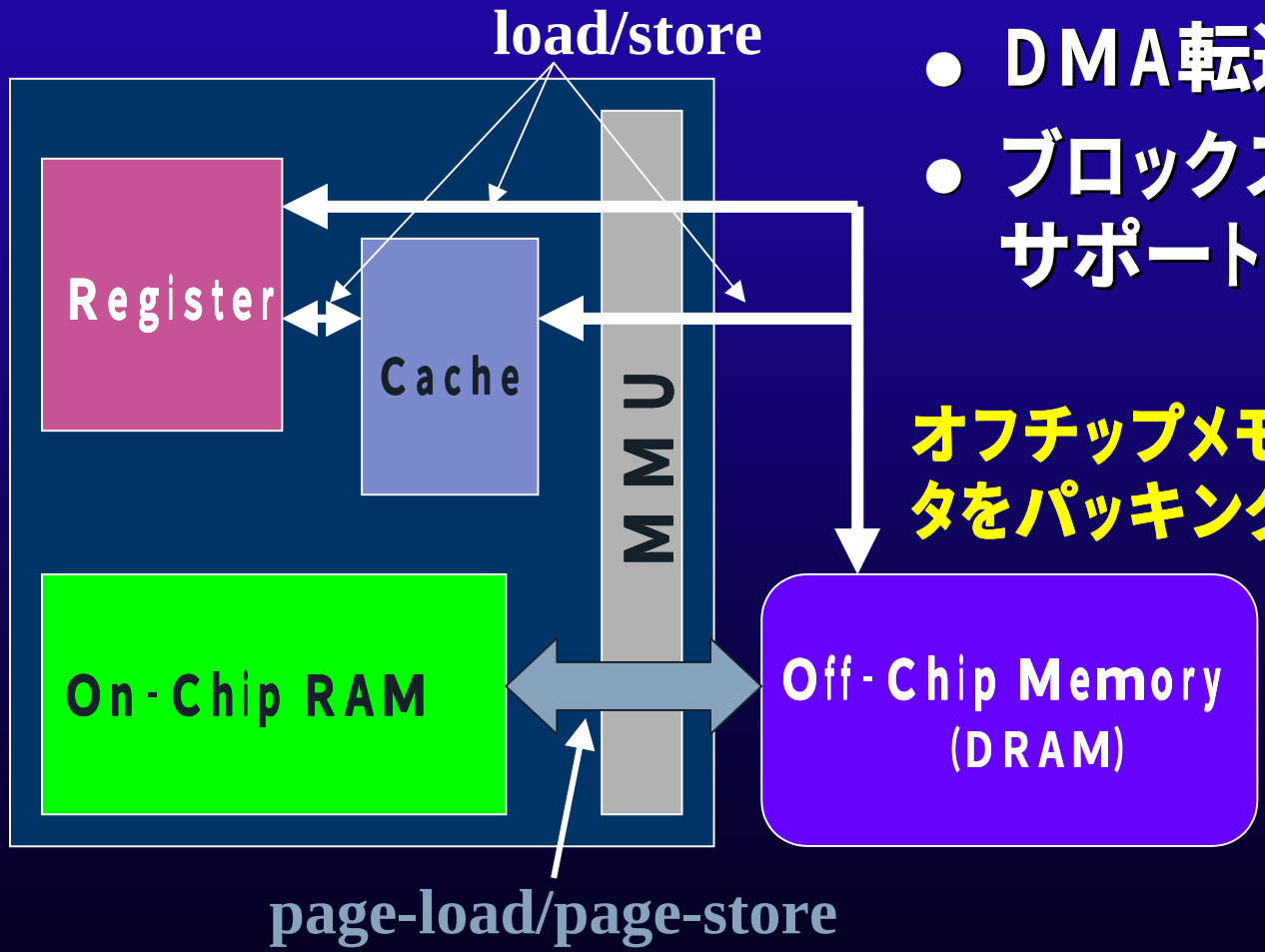
- 特別なLargeページとして管理する
  - On-Chip Address Start Register (ASR)
  - On-Chip Address Mask Register (AMR)

ー ベースレジスタとマスクを用い連続領域を取り出す



追加・拡張命令（1）

- page-load／page-store　512B～数KB
  - オンチップ・オフチップメモリ間データ転送命令



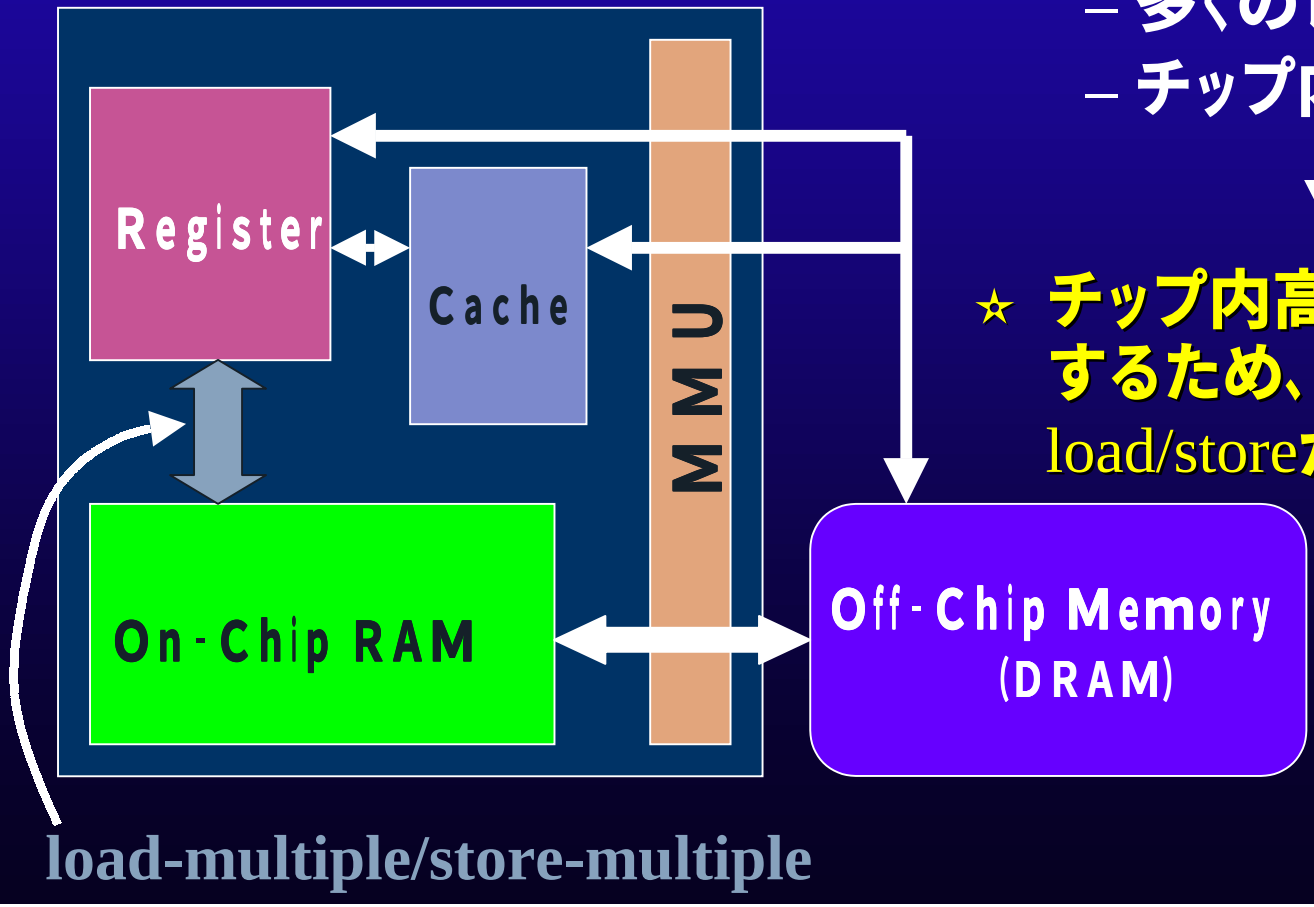
- DMA転送により実現
- ブロックストライド転送をサポート

↓

オフチップメモリ上の不連続データをパッキングしてオンチップに

追加・拡張命令（2）

- load-multiple／store-multiple
  - オンチップメモリ・レジスタ間データ転送命令
    - オンチップメモリ
    - 多くのレジスタ
    - チップ内高バンド幅



☆ チップ内高バンド幅を活用するため、複数wordの同時load/storeが必要

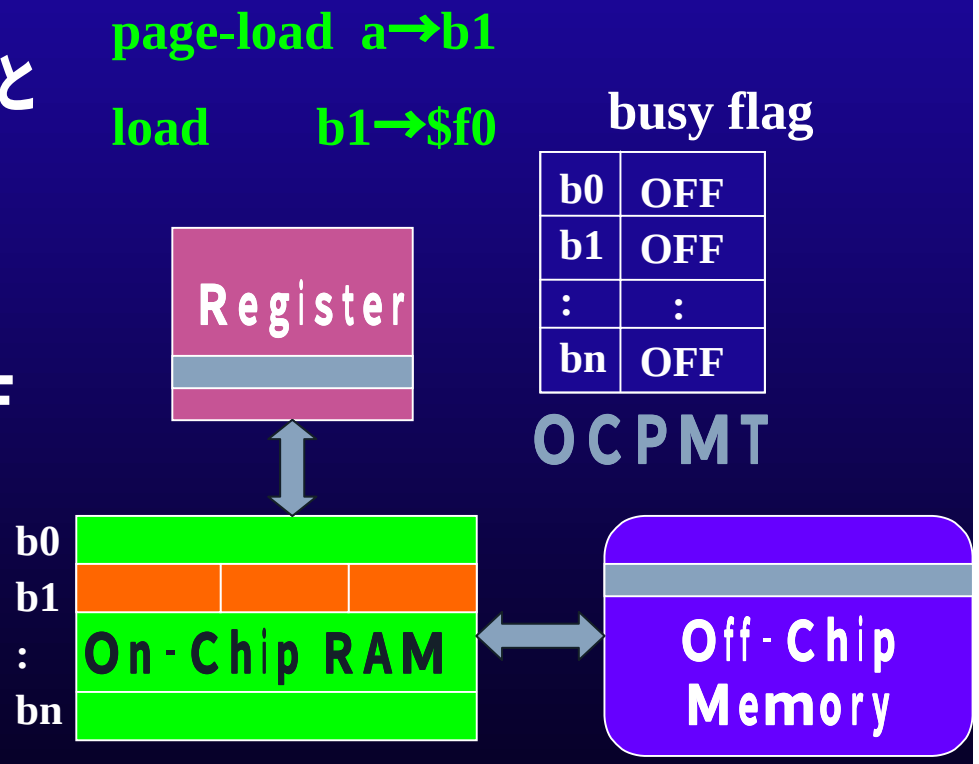
順序保証方法（1）

[A] page-load/storeの後でload/store

- *On-Chip Page Management Table*（OCPMT）と呼ぶ順序保証管理用のハードウェアを設ける

page-load/storeが発行されると

1. 該当pageのbusy flagをON
2. 該当オンチップメモリ領域へのload/storeはbusy flagがOFFになるまで待つ
3. オンチップメモリアクセス



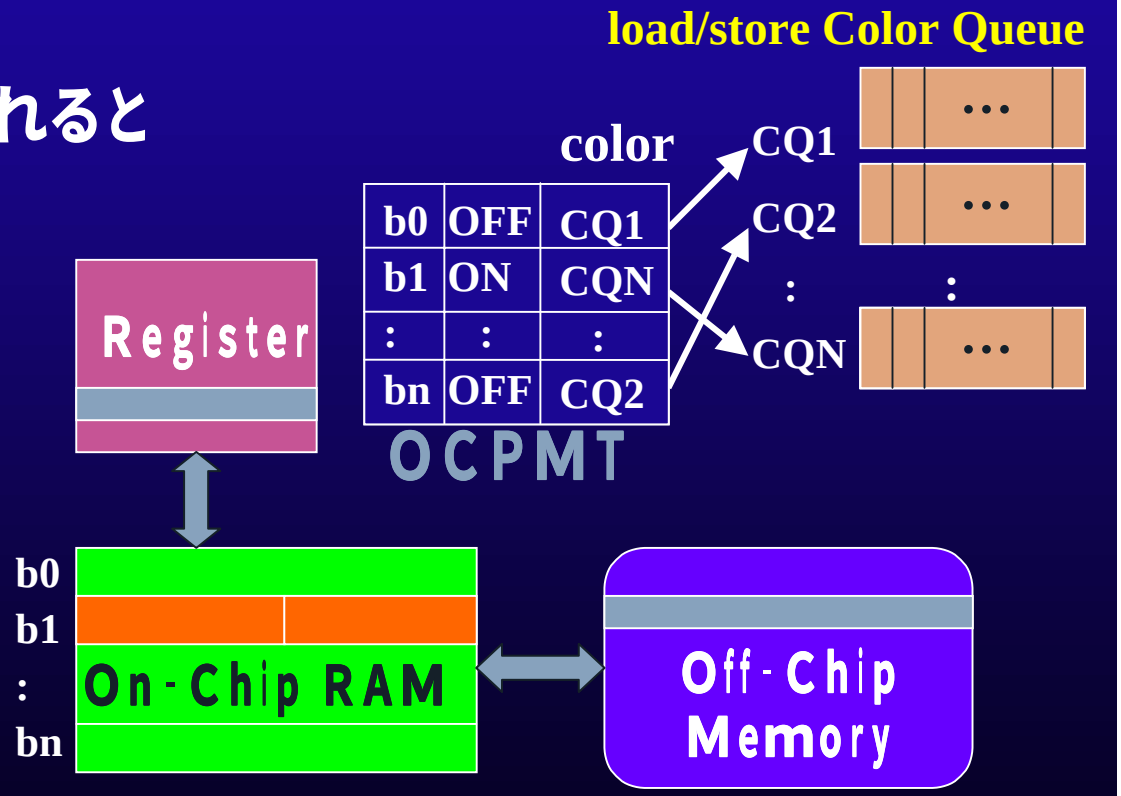
順序保証方法（2）

[B] load/storeの後でpage-load/store

- load/storeとpage-load/storeにcolorを設け、page命令はそれ以前に発行された同じcolorを持つload/storeの終了を待つ

page-load/storeが発行されると

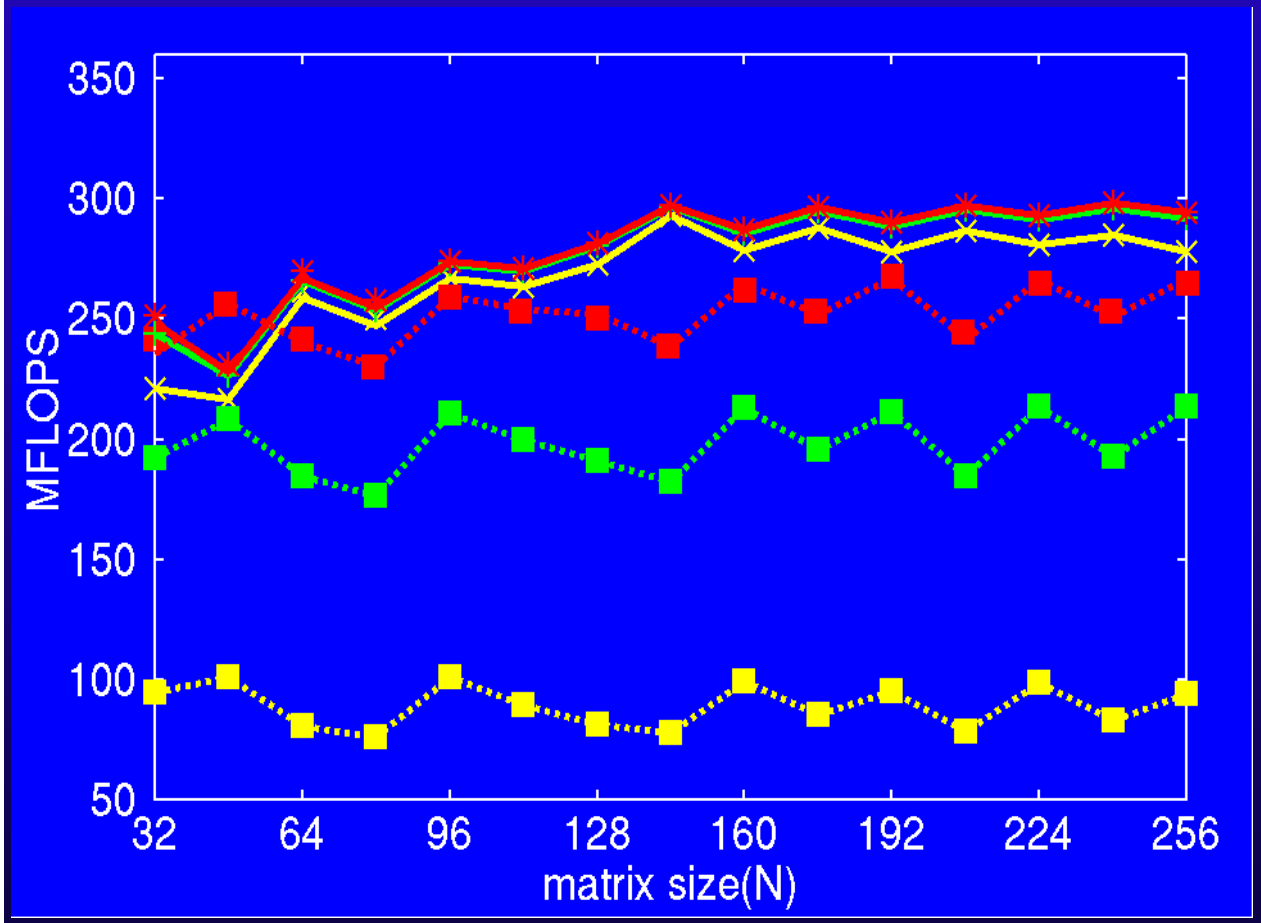
1. 該当colorをもつload/storeの終了を待つ（CQが空になるのを待つ）
2. 該当pageのbusy flagをON
3. page命令を実行



行列積プログラムによる性能評価

- SCIMA用コード
  - 36*36のブロックにオンチップメモリブロッキング
    - 配列A,B,Cのそれぞれのブロックをオンチップメモリに載せる
  - レジスタブロッキング
    - (i,j,k)=(4,4,1)でunrollingし、最内ループは8load,16madd
- キャッシュ用コード
  - 64*64のブロックにキャッシュブロッキング
    - 再利用性の最も高い1ブロックをキャッシュ上で有効利用
    - ブロック要素同士でコンフリクトを起こさないように連続領域にコピー
  - レジスタブロッキング
    - (i,j,k)=(4,4,1)でunrollingし、最内ループは8load,16madd

SCIMAとキャッシュの比較



SCIMA

オフチップメモリレイテンシ

0 cycle

10 cycle

60 cycle

キャッシュ

オフチップメモリレイテンシ

0 cycle

10 cycle

60 cycle

- SCIMAはキャッシュに比べ高性能
- キャッシュはオフチップメモリレーテンシの影響で性能が大きく低下

ソフトウェア制御による性能向上

SCIMA（レイテンシ　0cycle）　＞　キャッシュ（レイテンシ 0cycle）

SCIMAはキャッシュの1.1倍の性能

- キャッシュの場合は余分なメモリトラフィックが発生
  - コンフリクトによるキャッシュミス
  - ブロック要素を別の連続領域にコピー

⇒効率の良いデータのリプレースメント制御が実現

レイテンシによる性能低下の低減

SCIMA（レイテンシ　60cycle）　>>　キャッシュ（レイテンシ 60cycle）

SCIMAはキャッシュの3倍の性能

- SCIMAは大きい粒度のOn-Chip Page単位（1kB）にレイテンシ
- キャッシュは小さい粒度のライン単位（32B）にレイテンシ

⇒大きな粒度でオンチップ⇔オフチップ間のデータ転送を行うことにより実効的なスループットの確保

SCIMA（レイテンシ　60cycle）　＞　キャッシュ（レイテンシ　10cycle）

SCIMAはキャッシュの1.4倍の性能

⇒SCIMAでは二次キャッシュがなくても高性能

次世代超並列計算機へ

- **プロセッサ（例）**
  - SCIMA : 8 pipeline (MA) ✕ 2 GHz = 32 GFLOPS
  - **必要に応じて** 2〜8 proc. / node
  - 8 proc./node ✕ 512
    4 proc./node ✕ 1024
    ⇒ 130 TFLOPS
- **相互結合網**
  - **光インターコネクションの利用**
  - **粗粒度データ転送**
  - **通信レイテンシ隠蔽**
  - **厳しい演算性能／通信性能比をどう解決するか**

並列入出力・可視化システム

- 大規模科学技術計算の中間・最終結果の解釈
  - 数値データ⇒グラフィックス化
- 大量なデータの外部環境への引き出し
  - 外部との結合ネットワークがボトルネック
  - 並列に生成されたデータの逐次表現
    （例：独立な複数ファイルから単一ファイルへ）
- 外部の各種サーバ（ファイル・サーバ、グラフィックス・サーバ等）の並列化への対応

外部環境への並列データ転送＆可視化

並列入出力システムの要件

- 並列に生成されたデータをいかに並列のまま外部に引き出すか？（またはその逆）
- 並列計算機の並列入出力プロセッサの有効利用
- Commodity化している高速な汎用ネットワーク媒体の有効利用
- 広範なプラットフォームへの適用

並列入出力システム
(PIO: Parallel I/O System)

- Commodity ベースのネットワークインタフェース
  - 100base-TX, Gigabit Ethernet, etc.
  - 広範なプラットフォームへの対応
    (TCP/IP を用いた実装)
  - 安価で高性能なシステムの構築
- 並列入出力プロセッサを最大限に利用
  - 逐次化によるボトルネックのないシステム
  - 大容量バッファによる入出力処理と演算処理のパイプライン化
- アプリケーションレベルでの動的負荷分散
  - 分散メモリ型MPPの問題点をフロントエンドWSを用いて解決

ハードウェア構成

超並列計算機
CP-PACS
(2048 PU, 128 IOU)

並列ビジュアライゼーションサーバ
SGI Onyx2 (4 CPU)

AVS/Express

Switching
HUB
(multiple)

16

4

4

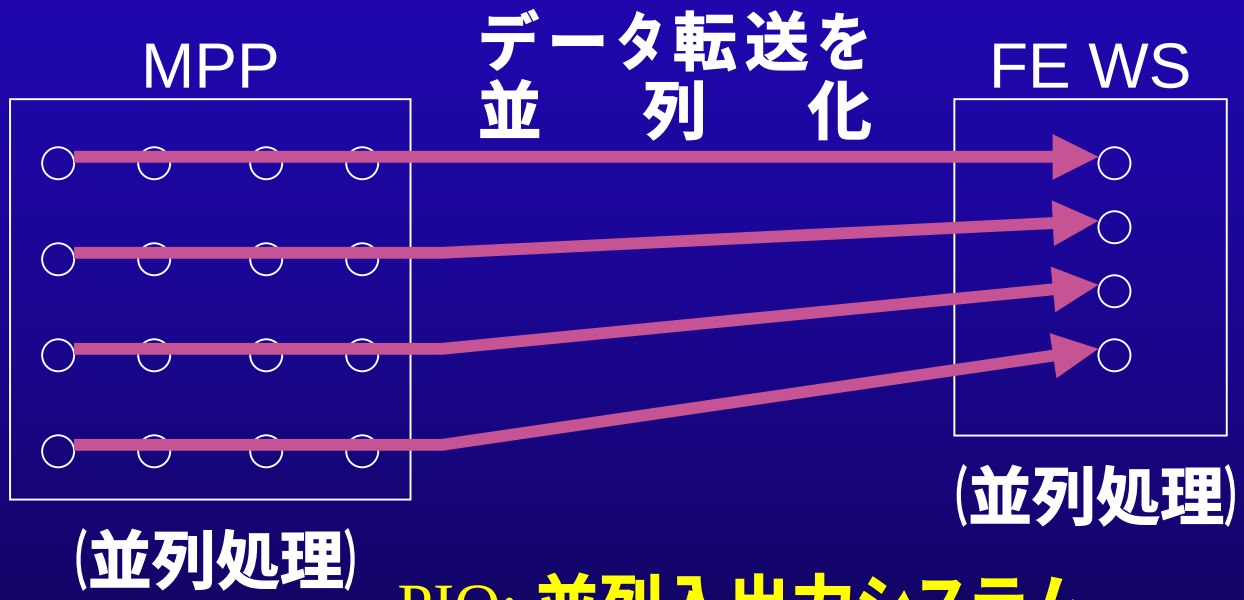
100Base-TX
Ethernet
による並列接続

並列ファイルサーバ
SGI Origin-2000 (8 CPU)

外部へ
(Ethernet等)

PIO: **並列入出力システム**

並列入出力システムの概念



PIO: **並列入出力システム**

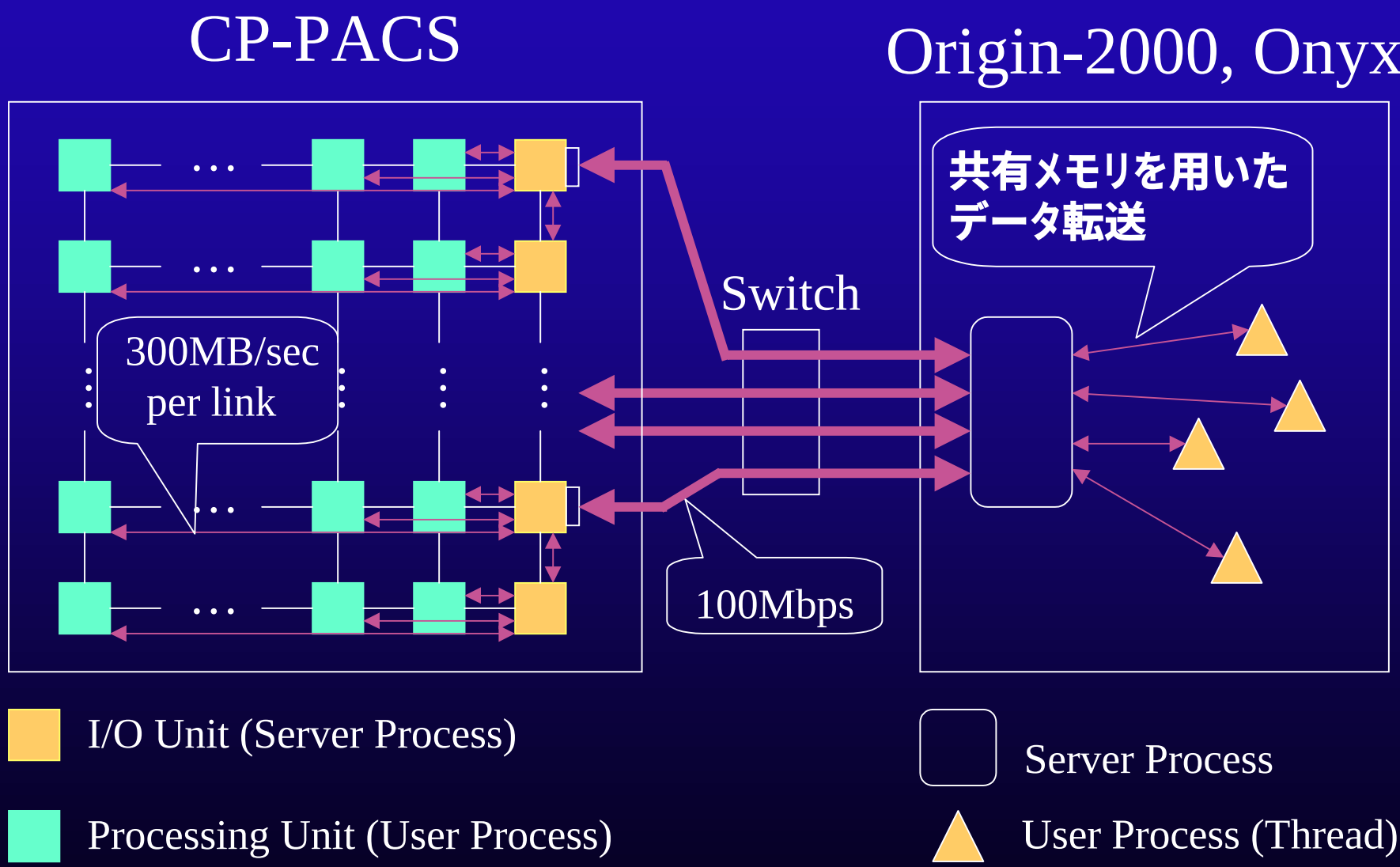
- アプリケーション・プログラムから外部のマシンへ直接データ転送
- チャネルの本数や負荷分散を特に意識する必要がない

並列入出力システムのソフトウェア構成

- TCP/IP stack 上に実装
  - ➔高い可搬性
- PIO システムの構成要素
  - 複数マシン上で動くユーザアプリケーション間のデータ通信・負荷分散を司る PIO Server
  - ユーザアプリケーションに対する API を提供するライブラリ群

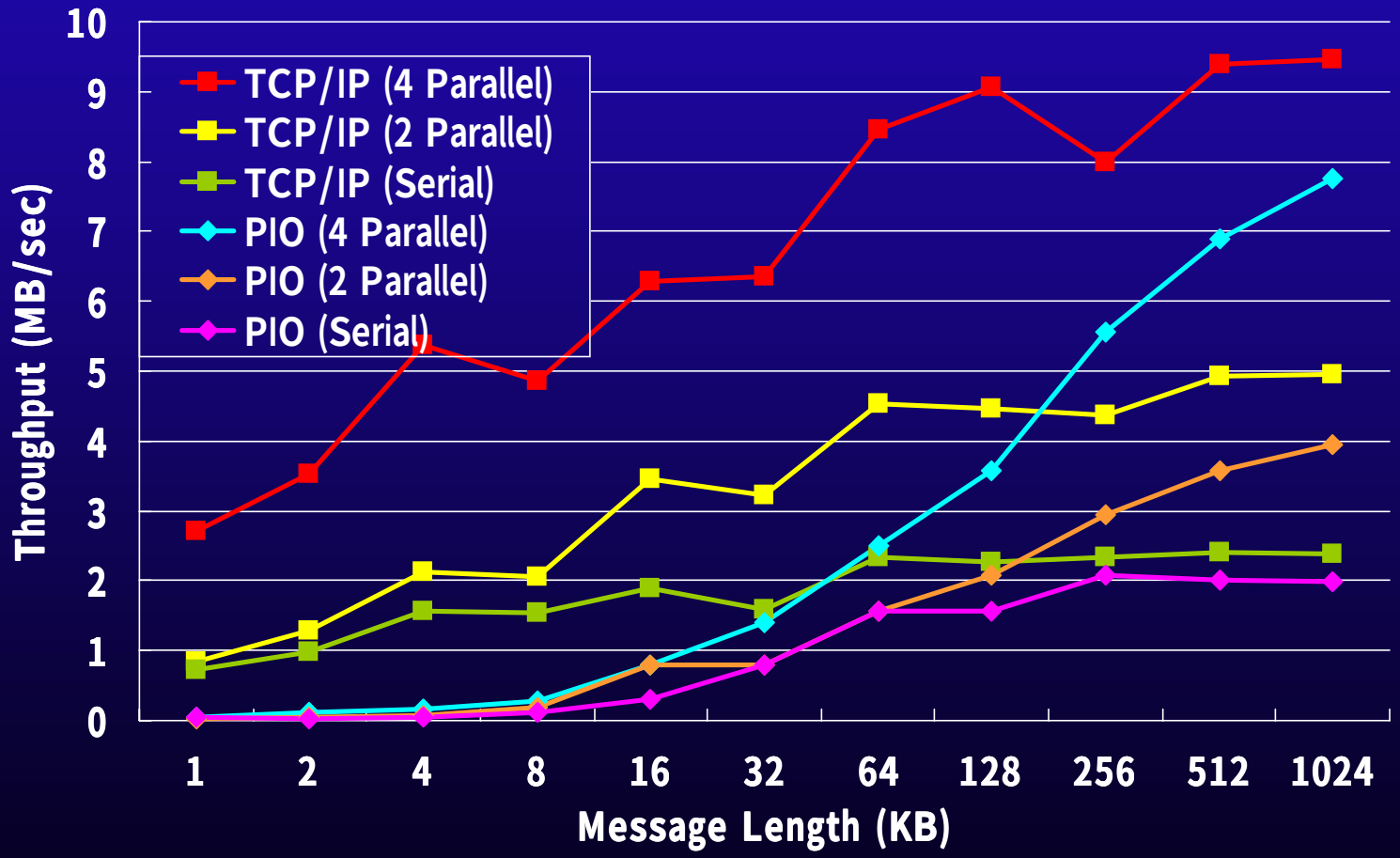
各ユーザプロセスは API を通して並列チャネルを意識せずに入出力処理を行う

並列入出力システムのモデル



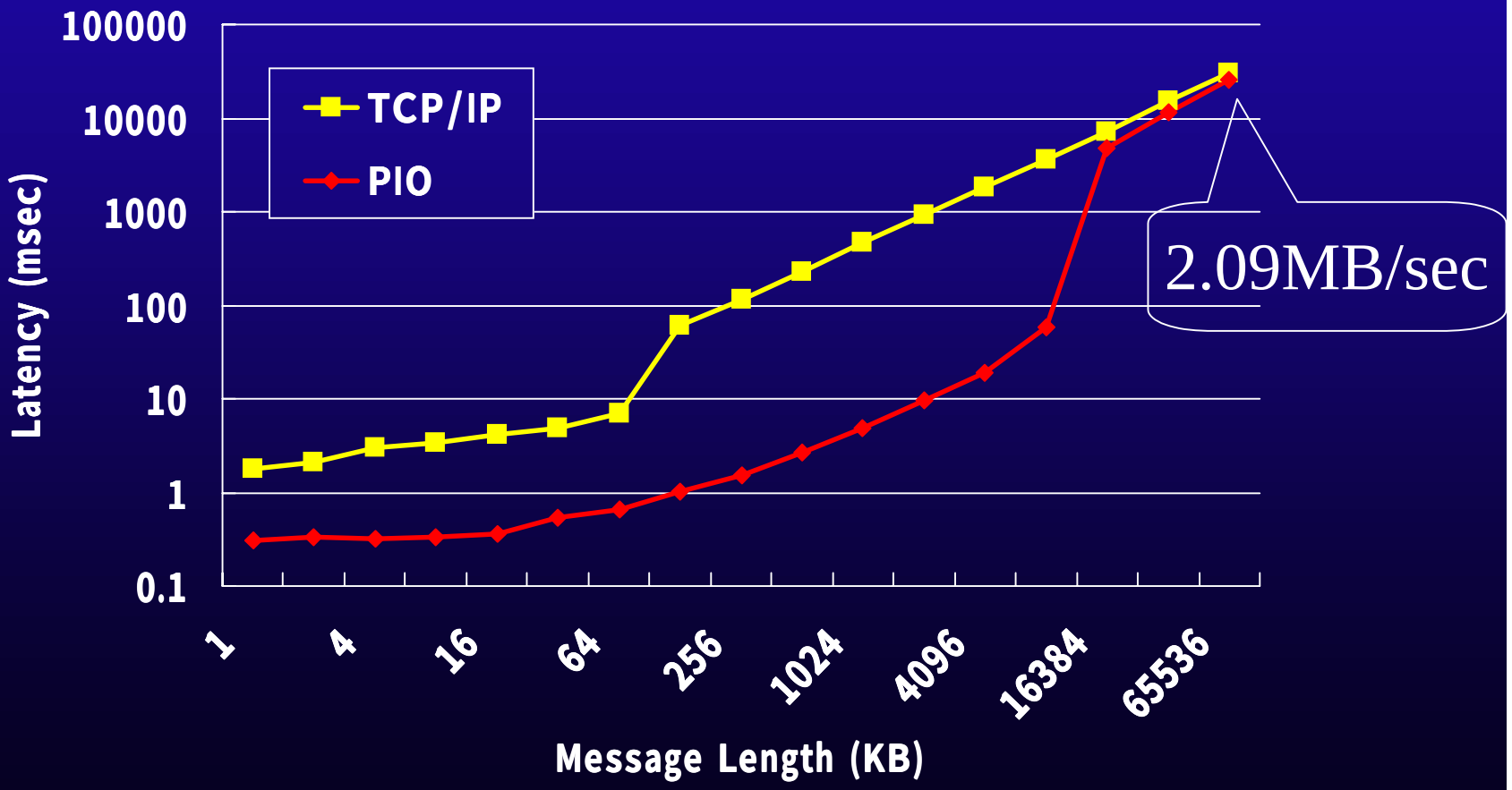
性能評価 — ping-pong 転送 —

CP-PACSと Origin-2000 (1-4 チャネル) 間の ping-pong 転送
(PIO と裸の TCP/IP の性能比較)



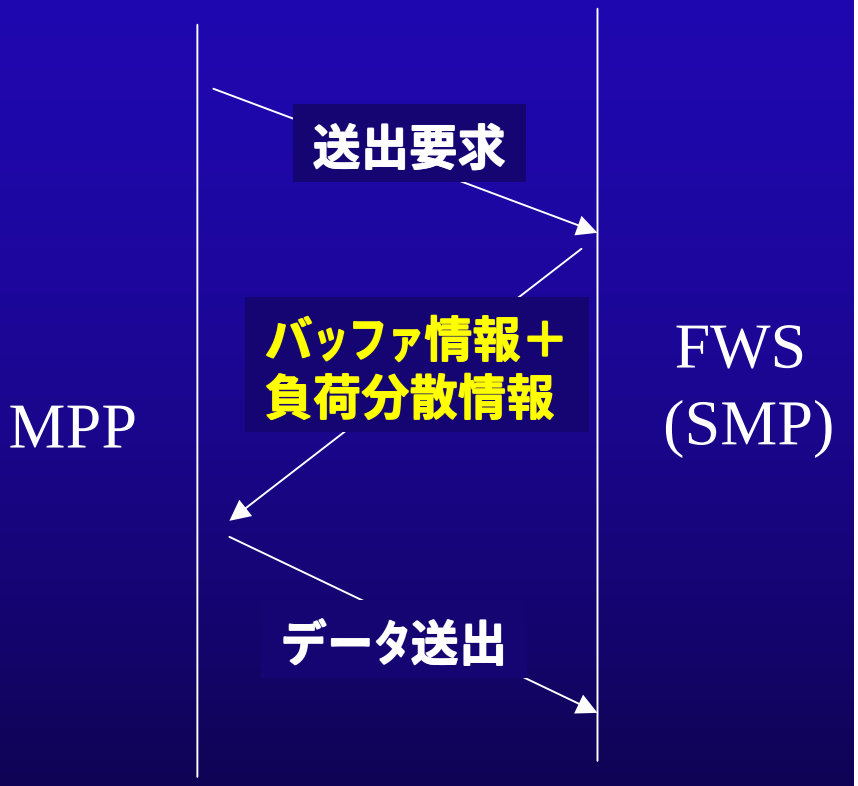
性能評価 ― 一方向転送 ―

CP-PACSとOrigin-2000 (1チャネル) 間の一方向転送においてアプリケーションプログラムに制御が戻るまでのレイテンシ



動的負荷分散

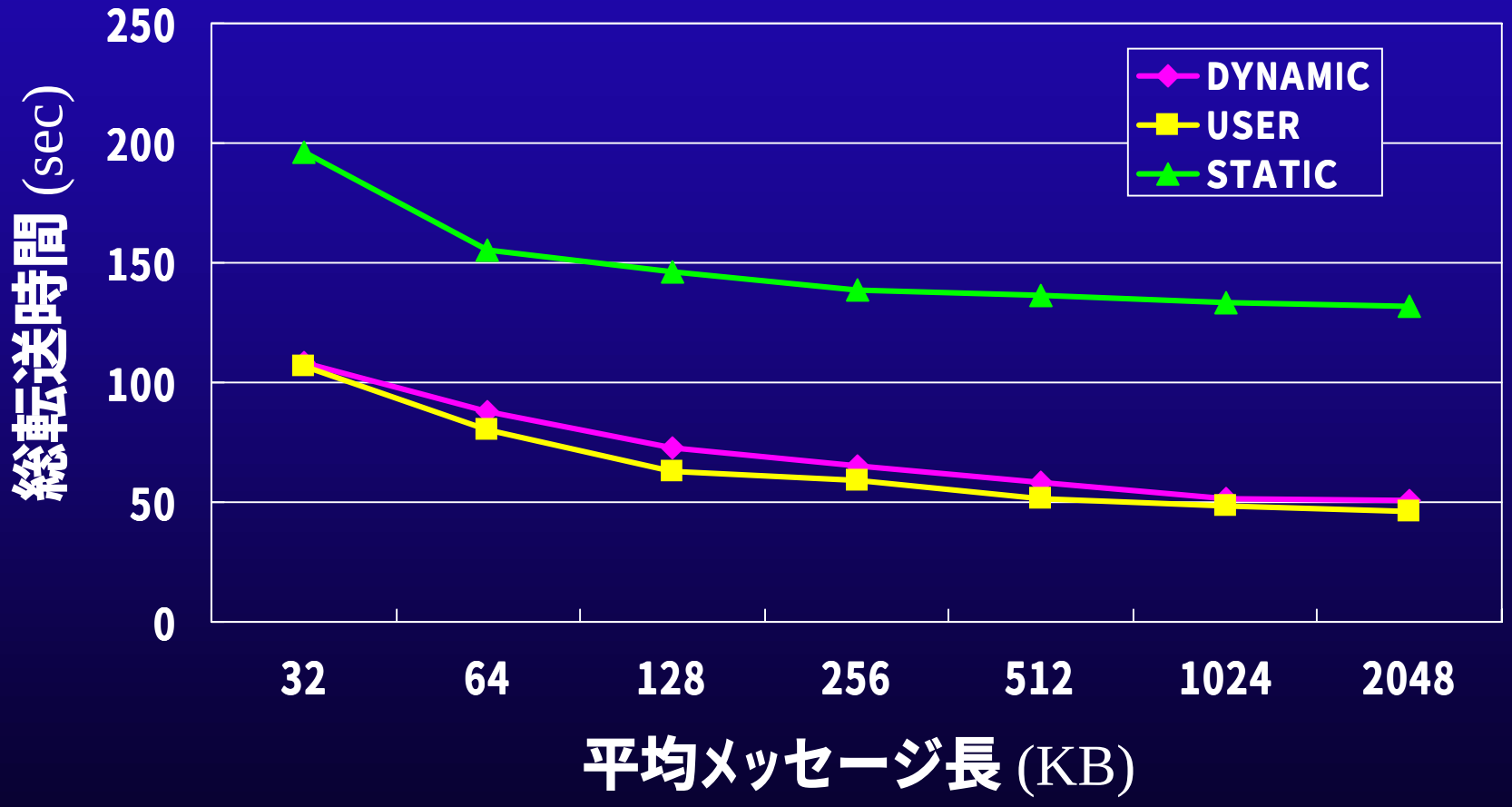
- MPP (分散メモリ型) における資源の動的負荷分散は、分散された情報管理のコストが大きい
- PIO では MPP と外部のフロントエンドワークステーション (共有メモリ) との協調動作を基本とする



フロントエンドWS上にMPPの負荷分散情報を保存し、
通信プロトコルに乗せてこれを適宜参照
⇒ 低コストで効率的な負荷情報共有が可能

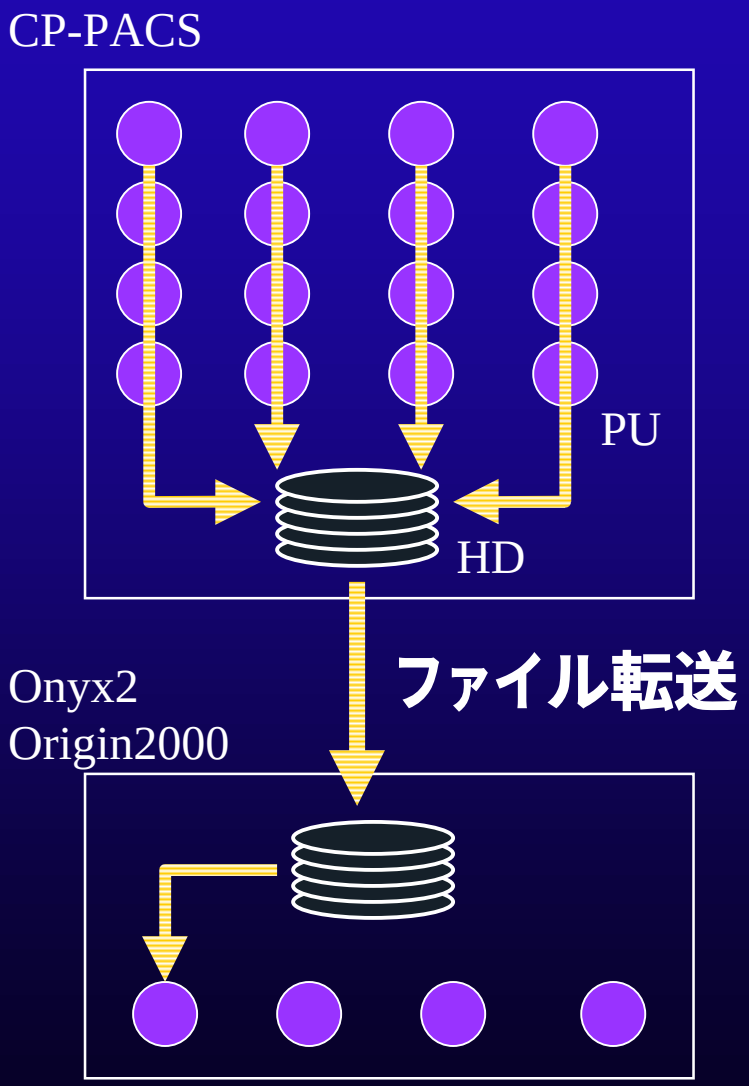
動的負荷分散機能の性能評価

最大16倍のばらつきを持った可変長メッセージの転送
(CP-PACS 256 PU を使用)



DYNAMICは、USERによる最適負荷分散とほぼ同一の性能

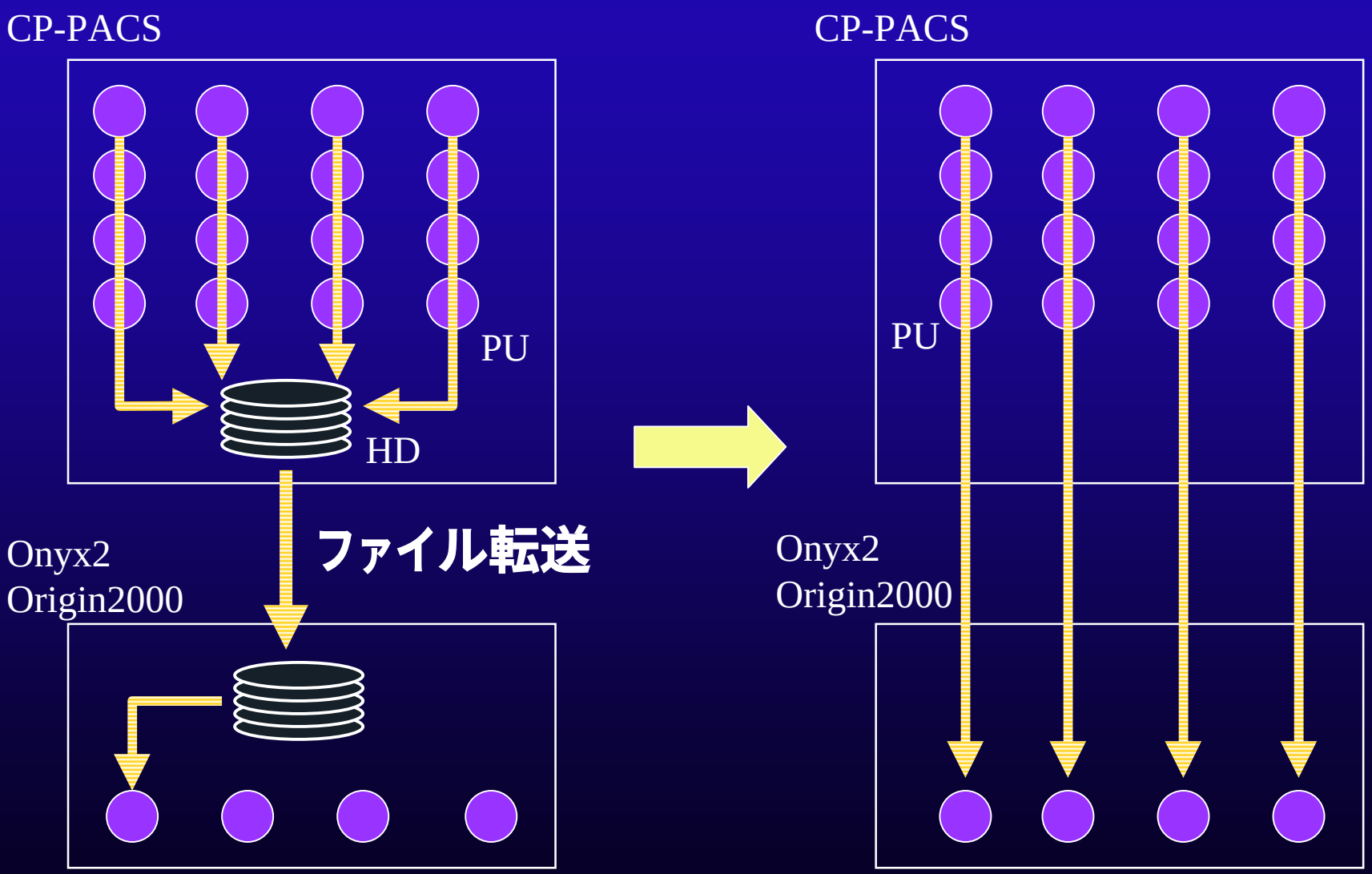
これまでの可視化手順とその問題点



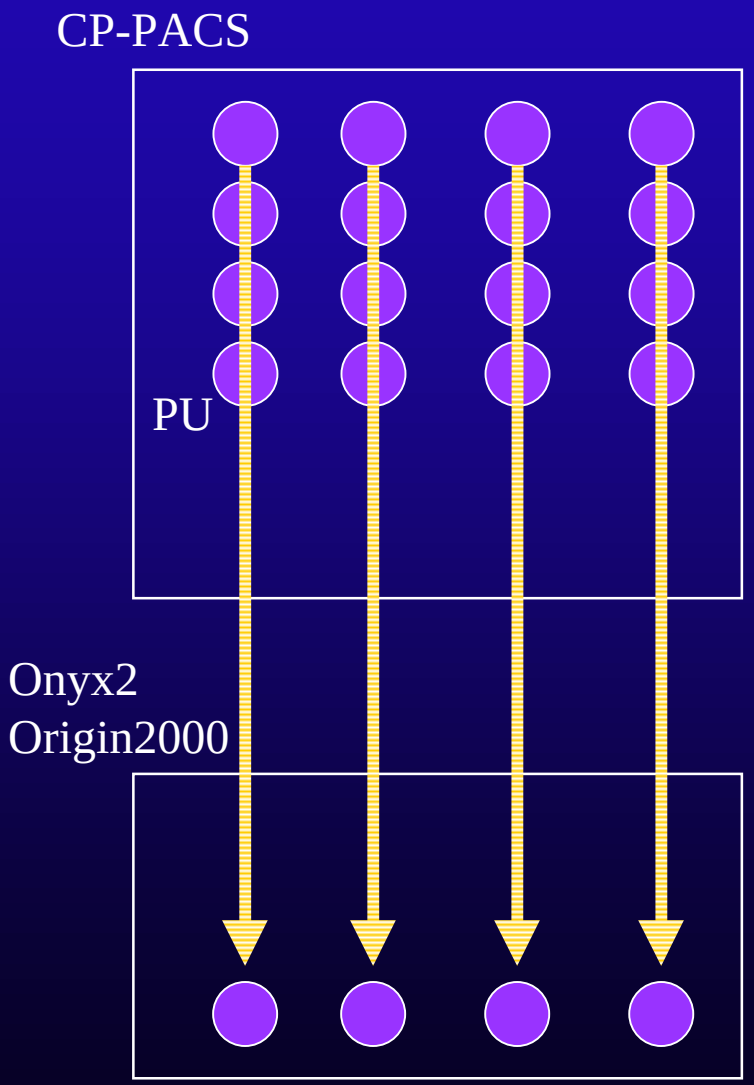
- 数値計算の終了後、結果を一括ファイル転送
- Onyx2で可視化

- 中間結果をモニターできない
- データが膨大であるため、ファイル転送や可視化処理が困難になる
- データを一度ひとつに束ねるため、ここで逐次処理が生じる

実装の方針（1/2）


CP-PACS
PU
HD
ファイル転送
Onyx2
Origin2000
CP-PACS
PU
Onyx2
Origin2000

実装の方針（2/2）



- 数値計算と可視化を同時並行的に実行
- 中間ファイルを生成せずにOnyx2へデータ転送
- 中間結果を随時読み込み、連続的に可視化
- データ転送および可視化処理もできる限り並列化
- AVS/Expressを基本に構築

PIO + AVS/Express

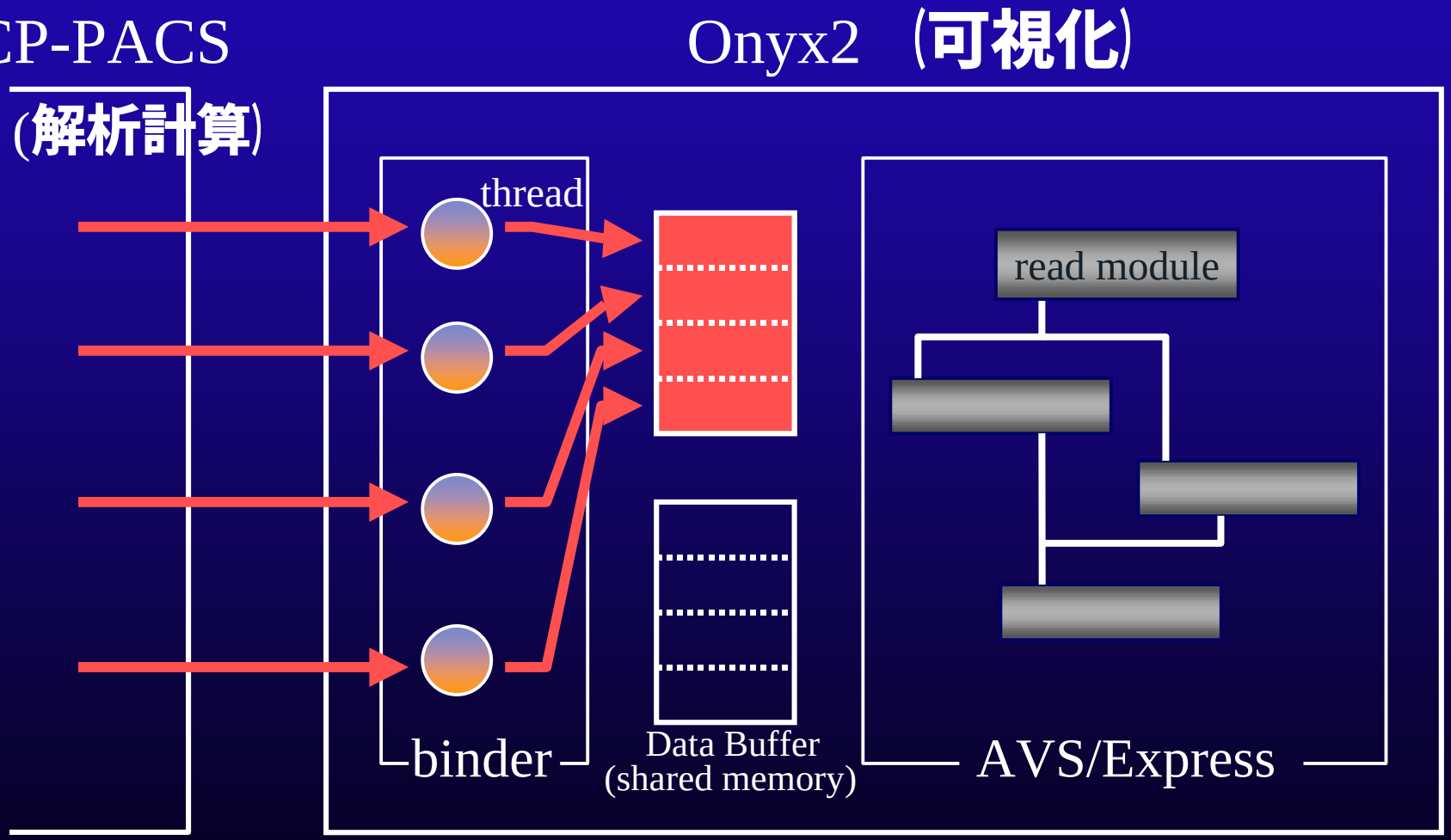
- 並列データ入力モジュール（リードモジュール）を実装

並列チャネルからのデータを直接読み込む

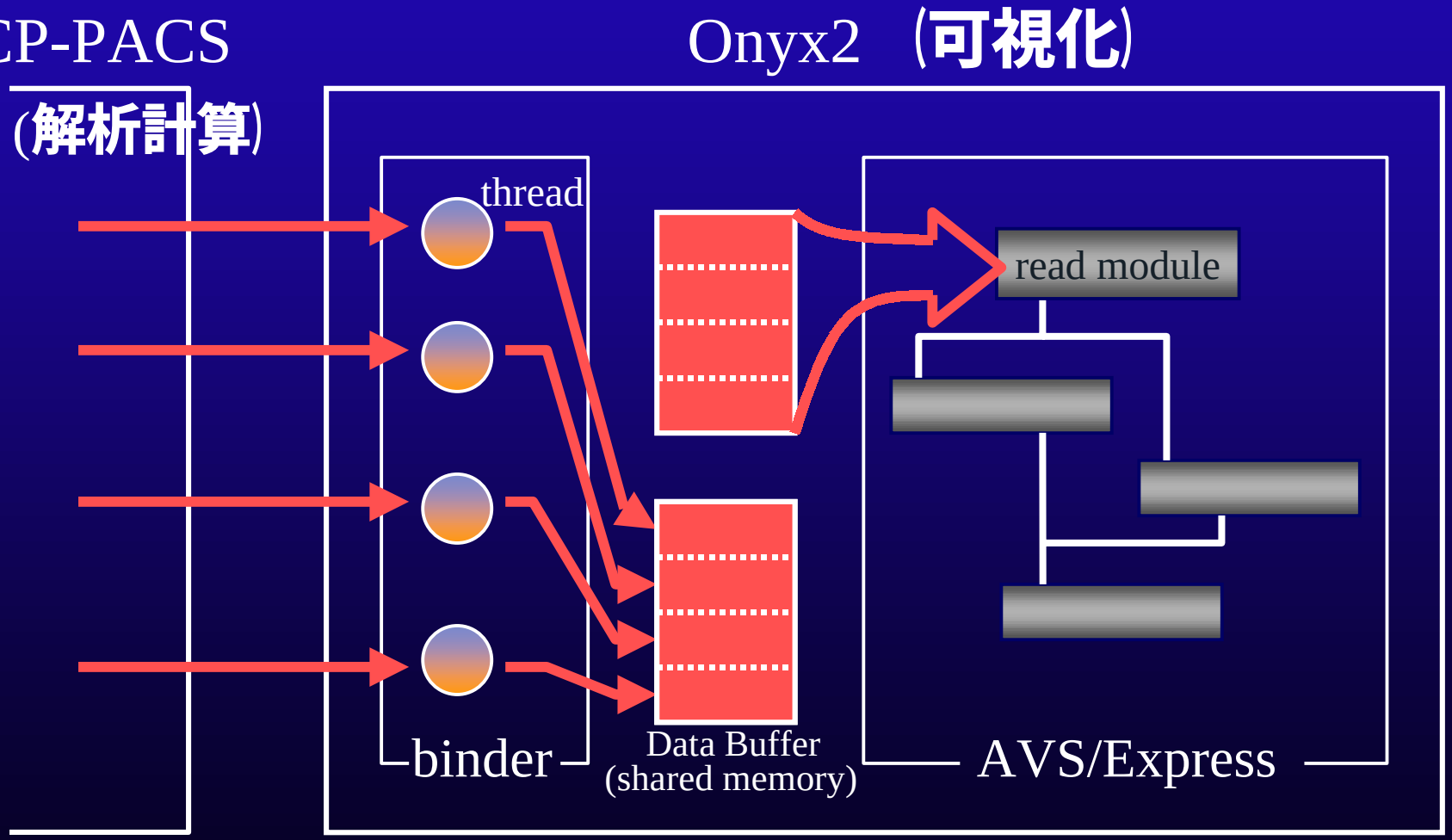
☆このままでは、データ入力と可視化処理が同時に行なえない

- データ入力部分を分離
  → 別プロセス（binder）として実装
- リードモジュールとbinderは共有メモリを介してデータの受け渡しをする

可視化までのデータの流れ



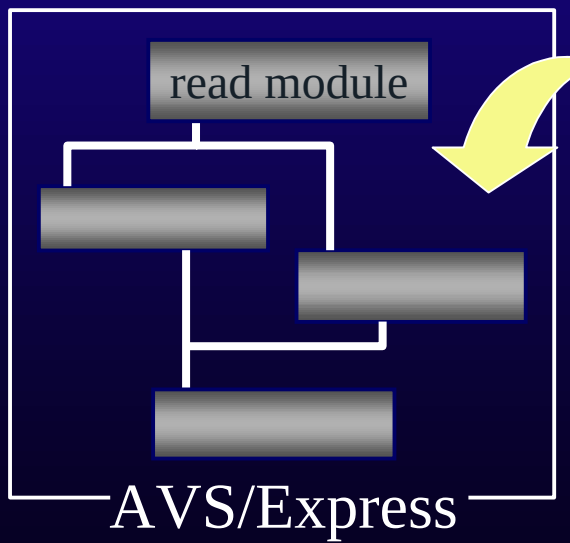
可視化までのデータの流れ



AVS/Expressへの並列モジュールの実装

- AVS/Express内部の処理はすべて逐次処理
- AVS/Expressの枠組みへ並列モジュールを実装

並列ボリュームレンダリング・モジュール



AVS/Expressへ並列モジュールを実装する

並列ボリュームレンダリング
モジュール

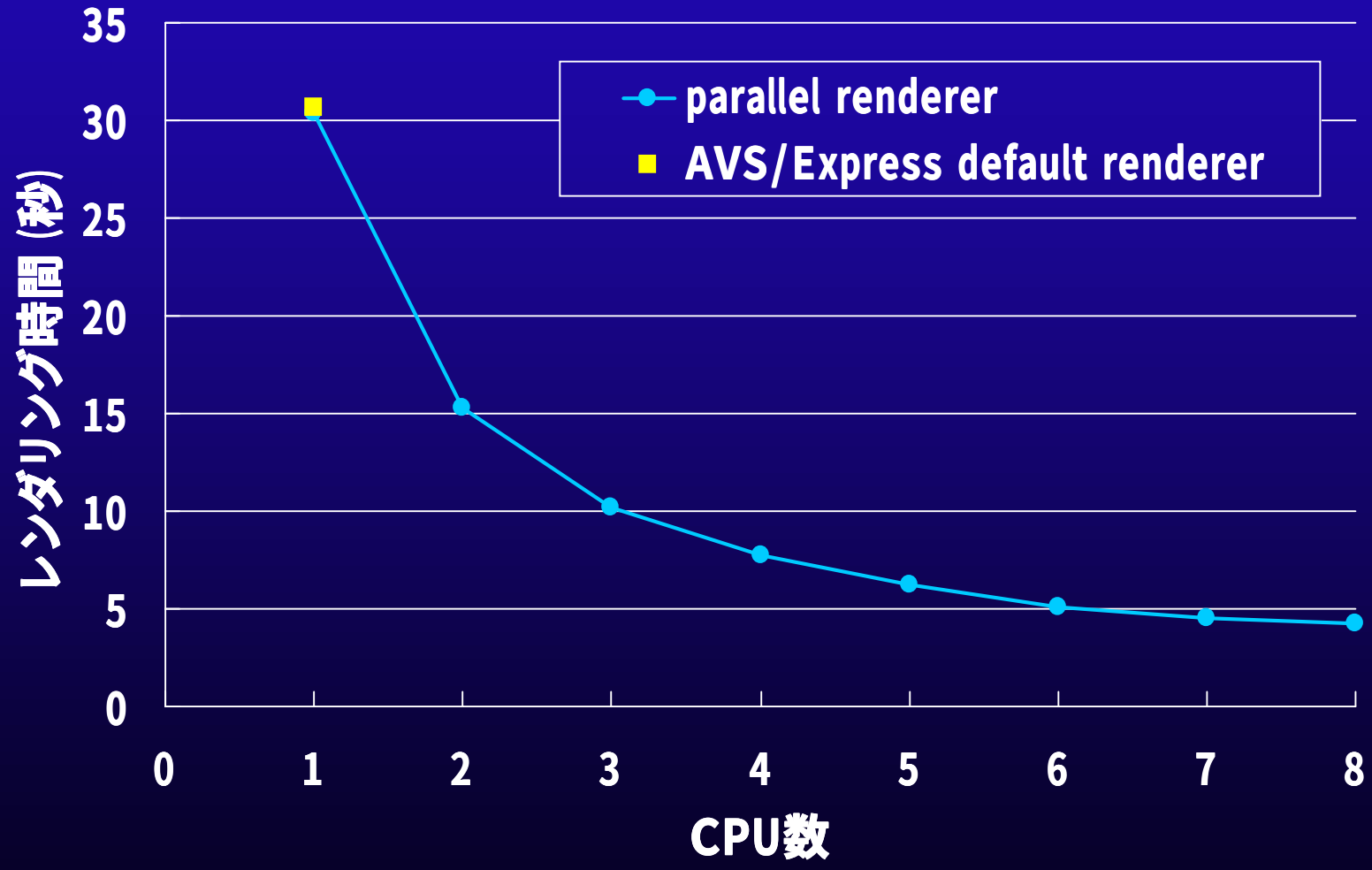
並列ボリュームレンダリング

- 並列ボリュームレンダリングをAVS/Expressのモジュールとして実装
  - アルゴリズムはレイキャスティング
  - pthreadを使い、スクリーンのスキャンラインをサイクリックにスレッドに割り当てる

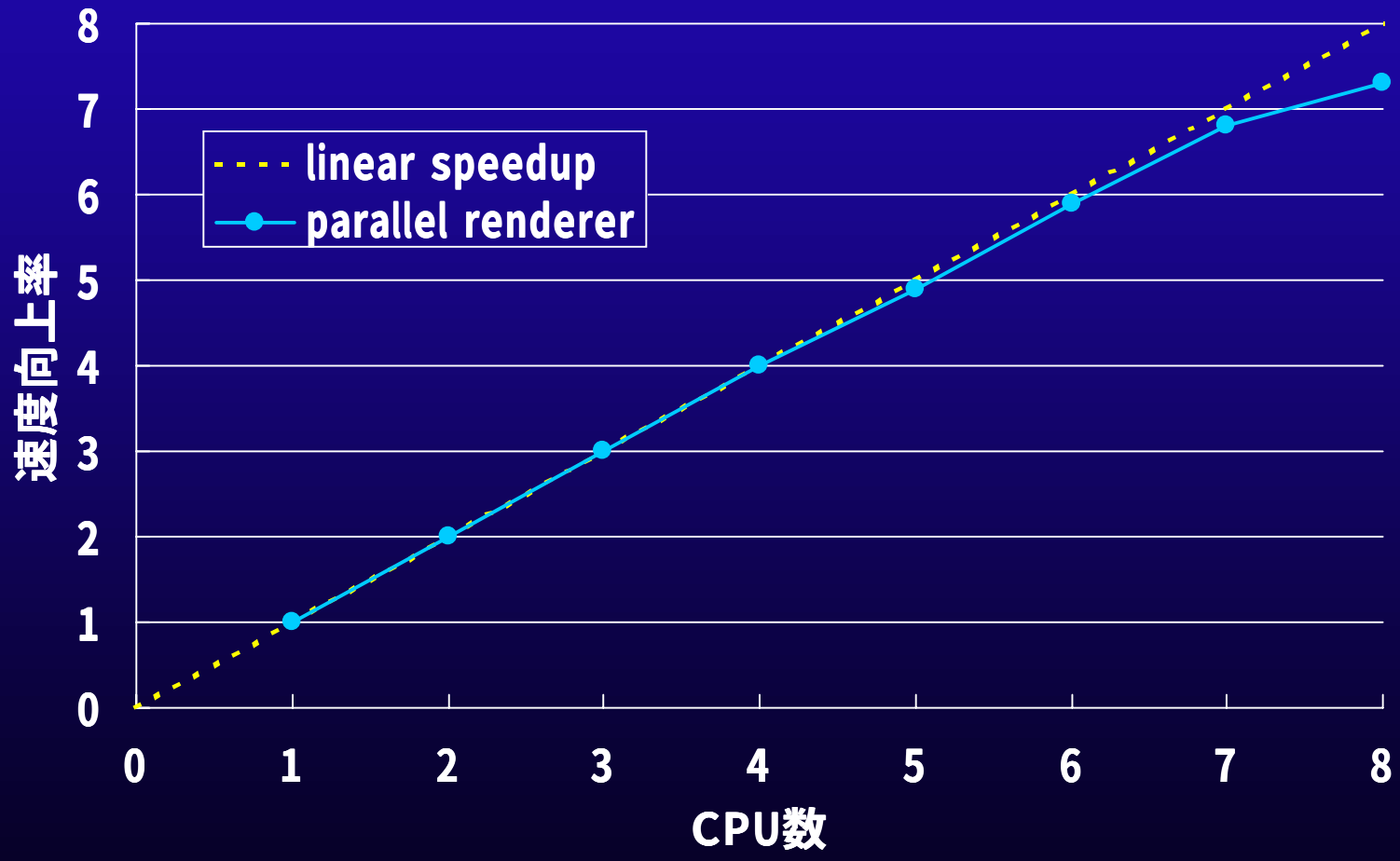
通常モジュールと同様な操作が可能

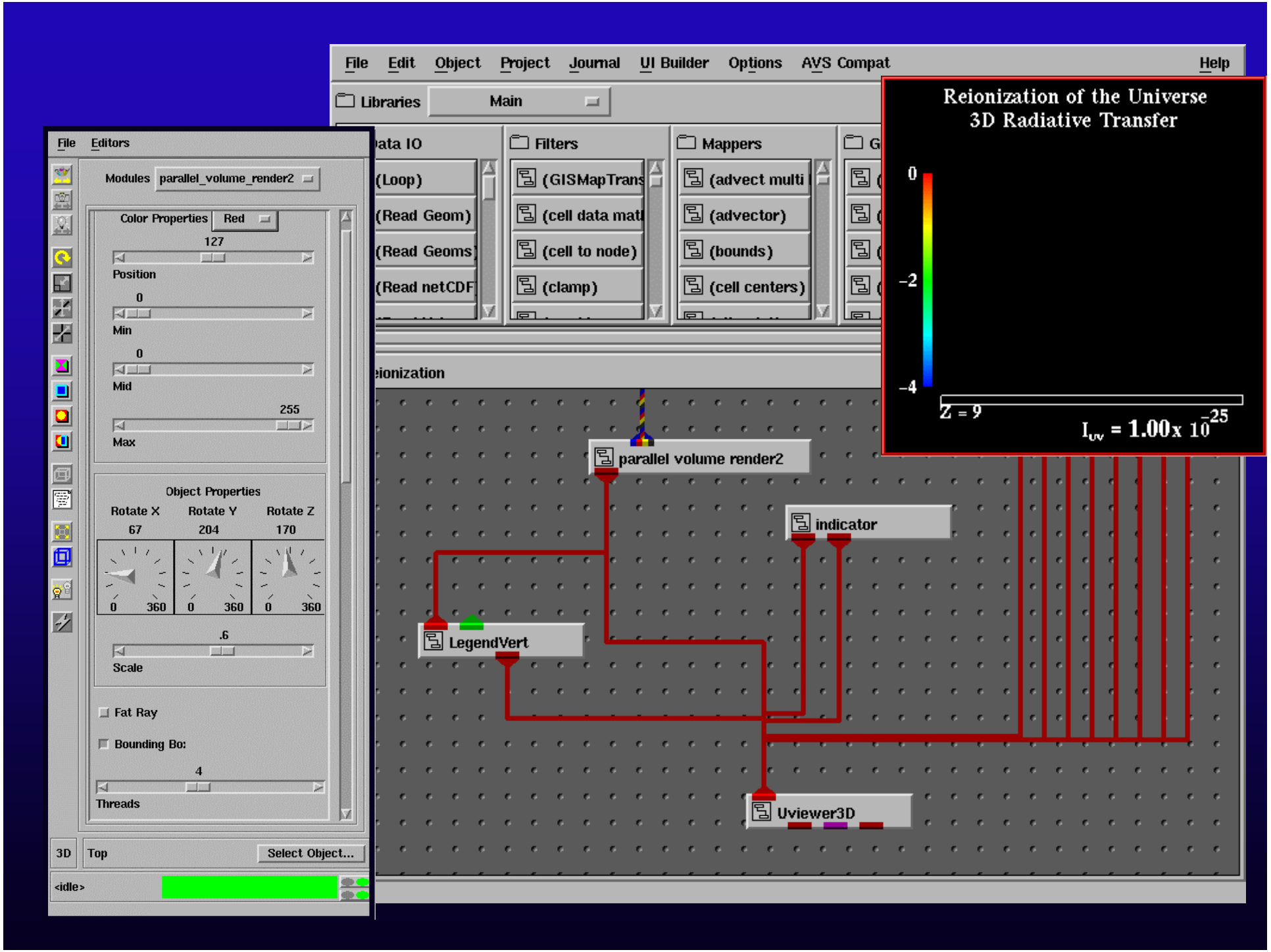
- 標準のボリュームレンダリング・モジュールとの比較
  - 128^3 格子データ（宇宙初期の水素ガスの電離状態）
  - スクリーンサイズは500×500
  - Origin2000で測定

評価結果（レンダリング時間）

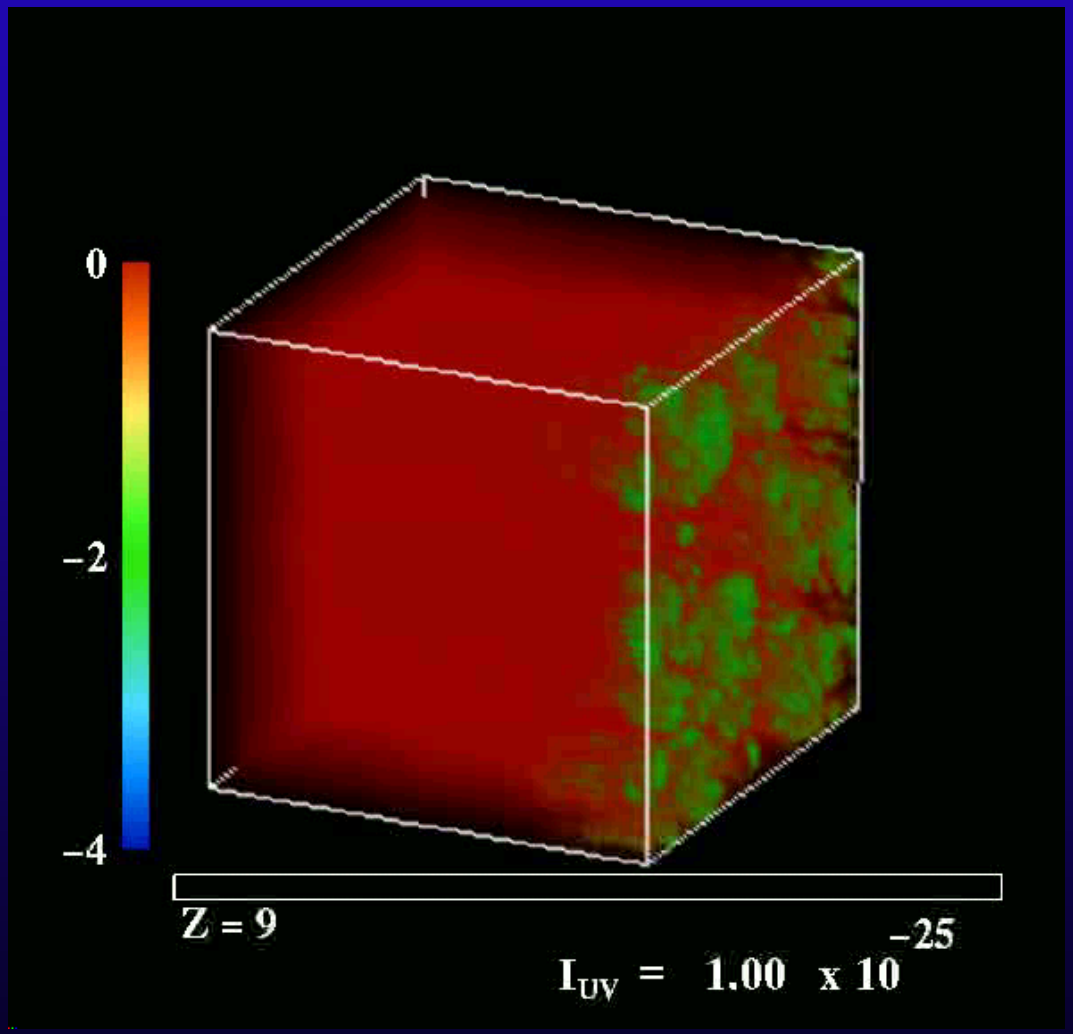


評価結果（速度向上率）


8
7
6
5
4
3
2
1
0
速度向上率
linear speedup
parallel renderer
0
1
2
3
4
5
6
7
8
CPU数


File Edit Object Project Journal UI Builder Options AVS Compat Help
Libraries
Main
Filters
Mappers
(Loop)
(Read Geom)
(Read Geoms)
(Read netCDF
(GISMapTrans
(cell data mat
(cell to node)
(clamp)
(advect multi
(advector)
(bounds)
(cell centers)
File Editors
Modules parallel_volume_render2
Color Properties Red
127
Position
0
Min
0
Mid
255
Max
Object Properties
Rotate X
Rotate Y
Rotate Z
67
204
170
0 360
0 360
0 360
.6
Scale
Fat Ray
Bounding Bo:
4
Threads
3D
Top
Select Object...
<idle>
Reionization of the Universe
3D Radiative Transfer
0
-2
-4
Z = 9
I_{uv} = 1.00x 10^{-25}
parallel volume render2
indicator
LegendVert
Uviewer3D

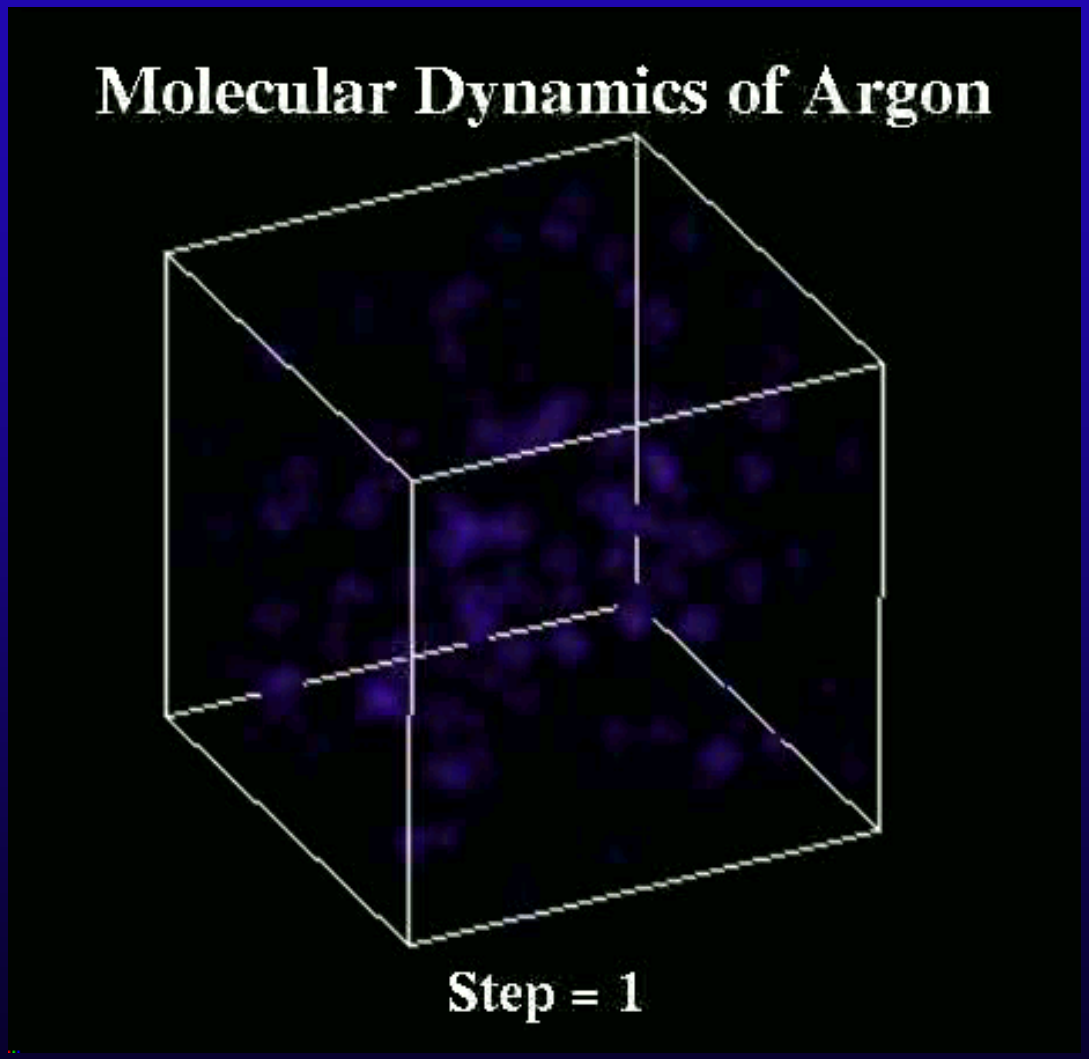
並列ボリュームレンダリング出力例（1）



宇宙の再イオン化のシミュレーション（単方向からの輻射）

2048 PUによる計算結果をPIO+並列可視化システムで処理

並列ボリュームレンダリング出力例（2）



液体アルゴンの分子動力学法シミュレーション

2048 PUによる計算結果をPIO+並列可視化システムで処理

並列可視化統合環境 (実装中)

- 並列入出力＋並列可視化システムの統合化制御環境
- 並列ジョブの起動、結果の可視化、中間または最終形式のファイル格納、ムービー作成等をGUI により制御
- 素データ・レベルでのロールバック機能
- ファイルの所在とデータ転送経路の自動判別

平成１１年度の成果

- 高性能プロセッサアーキテクチャ
  - SCIMAの基本仕様・方式の提案
  - 拡張命令の方式及び使用法に関する検討
  - ISAレベル及びクロックレベル・シミュレータの実装
  - 簡易性能評価ツールの実装
  - カーネルレベルのベンチマーク／性能評価
  - 基本アプリケーション（QCD、宇宙物理、etc.）の性能評価（進行中）
  - コンパイラの基本構想

平成１１年度の成果

- 並列入出力・可視化システム
  - CP-PACS実機への実装
  - 動的負荷分散機能の実現
  - PIOモジュールのAVS/Expressへの組み込み
  - 並列ボリュームレンダリング機能の AVS/Expressへの組み込み
  - CP-PACS実機を用いた総合テスト
  - 統合化環境の実装 (進行中)

今後の研究計画

- プロセッサアーキテクチャ
  - IPレベルを目指した論理設計
  - SMP化等、実システムを想定した設計
  - 相互結合網を含む並列処理システム設計・評価
  - コンパイラの開発
- 並列入出力・可視化システム
  - 実アプリケーションによる統合環境の評価
  - 言語レベルでの組み込みを含む実用化
  - 他のAPIへの適用
  - MPP内並列入出力装置を含む総合システム