Überblick

- MPI versus PVM
  - Geschichte
  - Vergleich
- Eigenschaften von MP-Interfaces (anhand von MPI)
  - Kollektive Kommunikation
  - Kommunikationsmodus
  - Nichtblockierende Kommunikation
  - Abgeleitete Datentypen
  - Groups/Communicators
  - Logische Topologie
- Erfahrungen mit MPI
- MPI2 Erweiterungen/Erwartungen

Geschichte von MPI/PVM

MPI

- 1993-94: Entwicklung des MPI1-Standards
  (Kommitee aus 40 Experten)
  230 Seiten, 128 Funktionen
- 1995-97: Entwicklung des MPI2-Standards
  zusätzlich 362 Seiten, 120 Funktionen

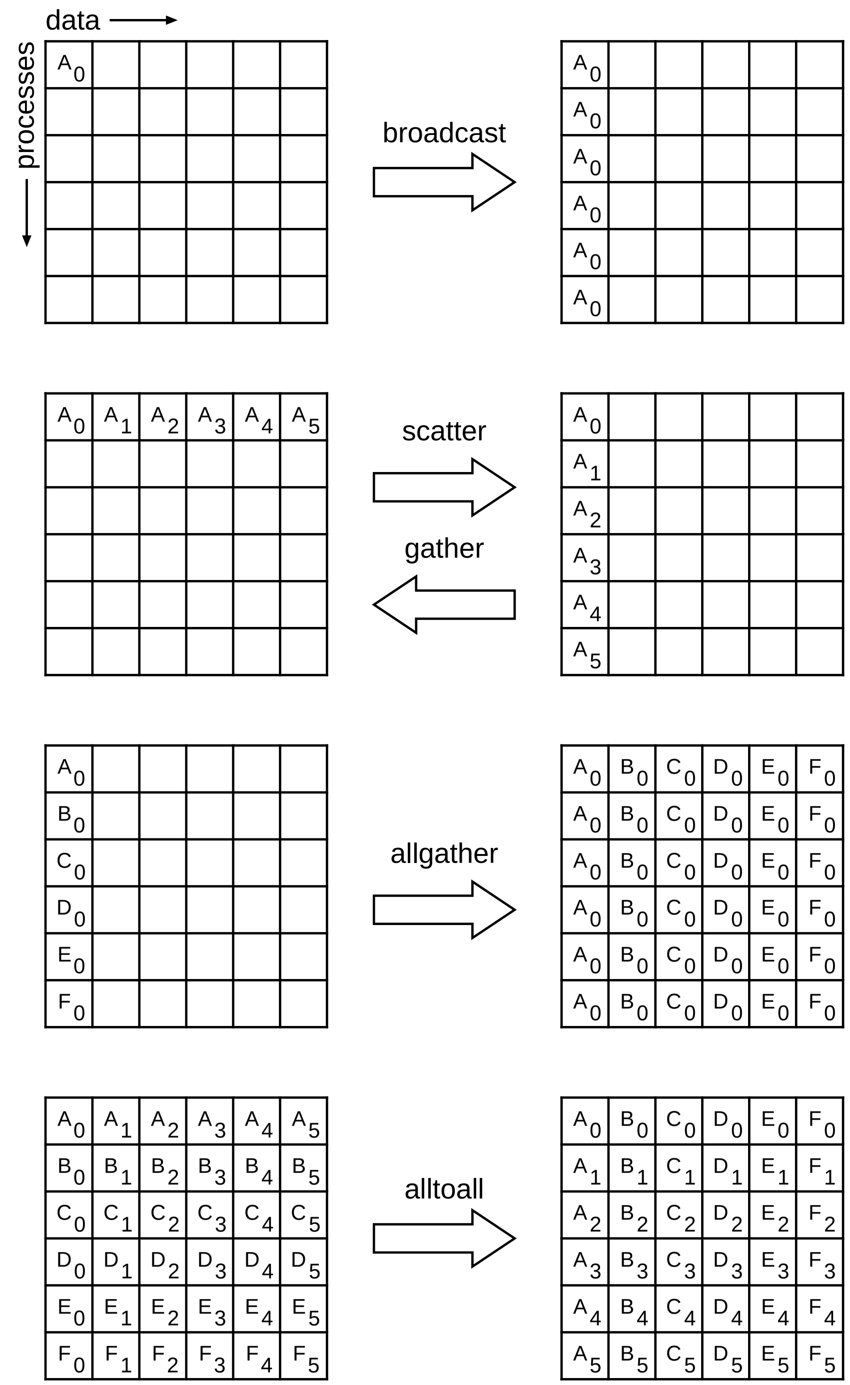
PVM

- 1989: Beginn der Entwicklung
  (V. Sunderam, Al Geist)
- 1991: Version 2.0: öffentlich verfügbar
  (J. Dongarra)
- 1993: Version 3.0: neues API, multiprozessorfähig

Vergleich von MPI/PVM

	MPI	PVM
Portabilität/Interoperabilität	versteht A-MPI B-MPI?	Ok, da 1 Implementierung
Prozeß-/Resourcenkontrolle	Prozeßkontrolle in MPI2	volle Virtual-Machine
Fehlertoleranz	Komplettes KILL	Benachrichtigung bei Fehler
Libraryfähigkeit	Communicator-Prinzip	Context in PVM 3.4
Asynchrone Kommunikation	voll	wenig
Topologieunterstützung	ja	kaum
Korrektheit	formal spezifiziert	Graubereich

Kollektive Kommunikation



Kommunikationsmodus

- Gepuffert (Send terminiert auch bevor Receive aufgerufen worden ist)
  - `MPI_BUFFER_ATTACH`: Zuweisung eines Sendepuffers
  - Puffer muß selbst alloziert werden
  - `MPI_BSEND`: Kopiert Daten in Puffer, wenn nicht sofort gesendet werden kann
  - Puffer ist kein! shared memory $\Longrightarrow$ Senderdaten $\rightarrow$ Puffer $\rightarrow$ shared memory $\rightarrow$ Empfänger
  - nicht für kollektive Kommunikation
- Synchron (`MPI_SSEND` terminiert erst wenn Receive aufgerufen worden ist)
- Ready (`MPI_RSEND` darf erst gestartet werden, wenn Receive wartet)

Nichtblockierende Kommunikation

- `MPI_ISEND`/`MPI_IRECV` terminiert sofort und returniert ein "request handle"
  - Datenbereich darf nicht verändert/benutzt werden, da noch nicht kopiert
- `MPI_WAIT(request handle)`
  - stellt sicher, daß Daten auf dem Weg sind
  - stellt NICHT sicher, daß Daten beim Empfänger angelangt sind (nicht einmal, daß Receive aufgerufen worden ist)
- Möglichkeit: `MPI_WAIT` erst wenn Datenbereich verändert werden muß ($\Rightarrow$ optimiertes aber unübersichtliches Programm)
- `MPI_WAITANY`, `MPI_WAITSOME` und `MPI_WAITALL` operieren auf Arrays von "request handles"

Abgeleitete Datentypen

- Aus bestehenden und vordefinierten Datenstrukturen neue Datenstrukturen ableiten
- Datentyp `MPI_DATATYPE` für alle MPI-Datentypen
- vordefiniert: `MPI_CHAR, MPI_INT, MPI_FLOAT, MPI_DOUBLE, ...`
- `MPI_TYPE_CONTIGUOUS`: Array aus Datentypen
- `MPI_TYPE_STRUCT`: Struktur aus Datentypen
- und noch viel mehr: `MPI_TYPE_VECTOR, MPI_TYPE_INDEXED, MPI_TYPE_HVECTOR, ...`
- Notwendig für nichtgepufferte Kommunikation

Groups/Communicators

- Groups
  - Jeder Prozeß besitzt innerhalb einer Gruppe eine Nummer (Rank)
  - Ranks sind durchgängig von 1 bis n
  - Der selbe Prozeß kann in unterschiedlichen Gruppen unterschiedliche Ranks haben
- Communicators
  - Communicator = Group + Context
  - Jeder Kommunikation ist ein Communicator zugewiesen
  - Communicator `MPI_COMM_WORLD` umfaßt alle Prozesse
  - `MPI_COMM_CREATE(comm,group,newcomm)` erzeugt innerhalb von `comm` einen neuen Communicator `newcomm` der die Prozesse aus `group` enthält
  - Alle Prozesse (aus `comm`) müssen `MPI_COMM_CREATE` aufrufen
  - `MPI_INTERCOMM_CREATE` erzeugt Intercommunicator (= Kommunikation zwischen zwei nichtüberlappenden Gruppen)
  - `MPI_COMM_DUP` erzeugt ein Duplikat eines Kommunikators (z.B. bei Initialisierung einer Library)

Logische Topologie

- Ziel
  - MPI kennt Topologie der Hardware.
  - Die Anwendung kennt die Topologie der Anwendung.
  - Die Anwendung teilt diese MPI mit.
  - MPI verteilt die Prozesse optimal auf die Prozessorelemente.
- Konstruktion einer Topologie
  - `MPI_CART_CREATE` erzeugt ein n-dim. kartesisches (möglicherweise periodisches) Gitter
  - `MPI_GRAPH_CREATE` erzeugt eine allg. Topologie aufgrund eines Graphen
  - Funktionen wie `MPI_CART_SHIFT` vereinfachen häufig benutzte Regeln

MPI2 Erweiterungen/Erwartungen

- Prozeß Management
  - `MPI_COMM_SPAWN` returniert einen Intercommunicator
  - `MPI_OPEN_PORT`, `MPI_COMM_ACCEPT` für Serverfunktionalität
- Einseitige Kommunikation
  - `MPI_WIN_CREATE` erzeugt ein Fenster durch das andere Prozesse auf prozeßeigene Daten zugreifen können
  - `MPI_PUT`, `MPI_GET`: Zugriff auf “remote memory”
  - `MPI_WIN_LOCK`: Locks auf Fenster erzeugen
- Erweiterte kollektive Kommunikation
  - auch für Intercommunicators
- Parallele I/O
  - `MPI_FILE_OPEN` eröffnet eine MPI-Datei
  - Views auf Teile der Datei
  - Individual und Shared File-Pointer
- C++ Anpassung