Jak wygląda praca na klastrze

Upraszczając nieco sprawę można powiedzieć, że klaster to dużo niezależnych komputerów (jednostek) połączonych mniej lub bardziej sprawną siecią. Często poszczególne jednostki połączone są (po kilka, kilkanaście ...) w węzły. Komunikacja przez sieć między jednostkami z jednego węzła jest szybsza niż między jednostkami z różnych węzłów.

Każda jednostka jest "pełnym komputerem" posiadającym swoją własną pamięć wewnętrzną i pełne możliwości samodzielnej pracy. Często pamięć dyskowa jest wspólna.

Klastry które znam, należą do ICM na UW. Pracowałem na starym HALO, na nowszym HALO2, na NOTOSIE (Blue Gene) i próbowałem na BOREASZU - trochę innym.

Wszystkie te maszyny są wyposażone w LINUX lub inny system operacyjny "linuxo-podobny" oraz w komplet najważniejszych kompilatorów (Fortran, C, ...). Ja piszę programy w moim ulubionym, (lekko podrasowanym) Fortranie 77.

Zwykle do wszystkich jednostek ładuje się ten sam program, który sam musi wiedzieć co ma w której jednostce robić.

Do wszystkich udostepnionych maszyn użytkownik dostaje się poprzez maszynę DELTA. Wejście na każdy "stopień" wymaga podania hasła. Najwygodniej pracuje się poprzez ssh i sftp, z dowolnego komputera podłączonego do publicznej sieci. W ten sposób nasz komputer staje się terminalem wybranej maszyny.

Gdy mamy już skompilowany nasz program, aby uruchomić go na klastrze, musimy program "wstawić do kolejki", gdzie czeka on na moment gdy zostanie wpuszczony do klastra i uruchomiony. Kolejka ma swoje dobre strony: gdy program jest w kolejce możemy, na przykład, pójść na spacer. Warto jednak, co pewien czas sprawdzać co w kolejce się dzieje.

Przed wstawieniem programu do kolejki, trzeba przygotować dla niego odpowiedni "script", to jest coś w rodzaju metryczki podającej różne niezbędne informacje o naszym programie. Taki "script" to rodzaj kilkulinijkowego formularza, który trzeba wypełnić i nadać mu nazwę, przy pomocy której nasz program jest identyfikowany. Przydział konkretnych procesorów następuje po odczytaniu przez system naszego "scriptu".

Do komunikacji między procesorami (jednostkami) służy system podprogramów MPI (Message Passing Interface). Te podprogramy są "wywoływalne" w sposób standardowy z programów pisanych w Fortranie i C. Podprogramy MPI uruchamiają operacje potrzebne przy komunikacji międzyprocesorowej. Przytoczę tu te najważniejsze, w wersji dla Fortranu, wywoływane komendą "call...".

Najczęściej korzysta się z "buforowanego" systemu przesyłania danych między procesorami. Bufor - to zarezerwowane miejsca pamięci służące do chwilowego przechowywania przesyłanych danych. Bufor trzeba zadeklarować na poczatku programu, przewidując dla niego

dostatecznie dużą liczbę miejsc. Używając bufora unikamy ewentualnego zablokowania programu, spowodowanego kolizją przy przesyłaniu. Poniżej podaję tylko "buforowe" komendy MPI dotyczące przesyłania.

1. **MPI_Init(ierr) otwiera środowisko MPI. Zmienna całkowita "ierr" zawiera informacje, czy operacja się powiodła**

2. **MPI_Comm_Size(MPI_Common_World, size, ierr)**

   zmienna całkowita "size" podaje liczbę procesorów które chcemy użyć; "MPI_Common_World" oznacza, że chcemy wybierać procesory z zasobu wszystkich dostępnych

3. **MPI_Comm_Rank(MPI_Common_World, rank, ierr); komenda ustala, że zmienna całkowita "rank" będzie oznaczac numer procesora, liczony od 0 do size-1**

4. **MPI_Finalize(ierr) zamyka środowisko MPI**

5. **MPI_Buffer_Attach(buffer,size,ierr),**

   MPI_Buffer_Detach(buffer,size,ierr)

   dołączanie i odłączanie bufora "buffer".

6. **MPI_Bsend(buf, count, datatype, dest, tag, Comm)**

   "buf" jest nazwą zbioru do wysłania, "count" oznacza liczbę wysyłanych słów danych,

   datatype:

   MPI_REAL, lub MPI_REAL8, lub MPI_Integer, ... "dest", to numer procesora dla którego przesyłka

jest przeznaczona, zmienna naturalna "tag", - to znacznik identyfikujący przesyłkę, "Comm", - to użyty tu skrót dla "MPI_Common_World"

7. **MPI_Recv(buf, count, datatype, source, tag, Comm, status, ierr)**

   "buf" jest nazwą zbioru, do którego przesyła się wysłane dane, "status" i "ierr" - te zamienne całkowite zawierają informacje na temat aktualnego stanu operacji przesyłania i odbierania

8. **MPI_Barrier(Comm, ierr)**

   każdy z wykonywanych procesów, czeka aż pozostałe procesy dotrą do tego miejsca

9. **MPI_Gather(sendbuf, sendcnt, sendtype, recvbuf, recvcnt, recvtype, root, Comm, ierr)**

   zbieranie ze wszystkich procesów; "sendbuf" - zbiór wysyłany, "recvbuf" - zbiór do którego się zbiera, "root" numer procesora w którym wszystko się zbiera

10. **MPI_Allgather(sendbuf, sendcnt, sendtype, recvbuf, recvcnt, recvtype, Comm, info)**

    to samo, co MPI_Gather(), tylko zbieranie we wszystkich procesorach będących w uzyciu

11. **MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, Comm, ierr)**

    przesyłanie "sendbuf" ze wszystkich procesów, do "recvbuf" w procesorze "root" i wykonywanie operacji "op". Operacja "op":

MPI_SUM, MPI_PROD, MPI_MIN, MPI_MAX i wiele różnych operacji logicznych i innych

12. **MPI_Allreduce(sendbuf, recvbuf, count, datatype, op, Comm, ierr)**

    to samo co MPI_Reduce, tylko przesyłanie do wszystkich używanych procesorów

Każda z maszyn ma nieco inny "język" scriptów, inny język porozumiewania sie w sprawie kolejki, sprawdzania stanu kolejki, wyciągania programów z kolejki itp.

Więcej informacji niż wyżej podane na temat MPI - patrz przeglądarka Google.