با توجه به گسترش روزافزون مجموعه‌ی .Net میکروسافت در بین برنامه‌نویسان و قابلیت‌های بسیار خوبی که در این مجموعه گنجانده شده است، بر آن شدم تا نحوه‌ی پیاده‌سازی روش MPI را در این مجموعه بررسی نموده و در قالب گزارشی کوتاه ارائه نمایم. اغلب مطالب این گزارش مختصر از سایت شرکت میکروسافت و مرجع [۱] اقتباس شده است. با امید به اینکه بتواند به عنوان نقطه‌ی شروعی در برنامه‌نویسی MPI با .Net باشد.

این گزارش به صورت زیر مرتب شده است در این قسمت به قابلیت های موجود در .Net جهت پیاده‌سازی MPI (MPI.Net[۱]) پرداخته می‌شود.

- ✓ نصب و راه‌اندازی MPI.Net
- ✓ نوشتن و اجرای یک برنامه‌ی ساده
- ✓ ارتباط دهنده‌ی[۲] MPI
- ✓ ارتباطات نقطه به نقطه[۳]
- ✓ ارتباطات گروهی[۴]
- ✓ ترکیب نتایج با موازی‌سازی

این راهنما به شما کمک می کند تا MPI.Net را بر روی ایستگاه‌های کاری چندنخی[۵] ویا بر روی سیستم‌های کلاستر نصب نموده و استفاده نمایید. همچنین به شما اجازه‌ی برنامه‌نویسی به کمک کلیه‌ی زبان‌های مجموعه‌ی .Net به خصوص C# را خواهد داد. MPI یک استاندارد جهت نوشتن برنامه‌های مبتنی بر ارسال پیام است. این استاندارد به طور گسترده‌ای در برنامه‌های موازی با کارایی بالا که بر روی کلاسترها و سوپر کامپیوترها اجرا می شوند، کاربرد دارد.

در سیستم‌های ارسال پیام پروسه‌های متفاوتی که بر روی سیستم‌ها بصورت موازی در حال اجرا هستند می‌توانند با ارسال پیام بر روی شبکه با همدیگر رابطه برقرار نمایند. بر خلاف سیستم های چندنخی که در آن نخ‌های متفاوت از حالت برنامه‌ی[۶] یکسانی برخوردارند، هر کدام از پروسه‌های MPI دارای حالت برنامه‌ی منحصر به فرد خود می باشند که این حالت، قابل تغییر و یا مشاهده توسط بقیه پروسه‌ها نمی‌باشد. بنابراین پروسه‌های MPI می‌توانند تا جایی که شبکه اجازه می دهد بر روی ماشین‌های متفاوت و یا حتی معماری‌های متفاوت در حال اجرا باشند.

اغلب برنامه‌های MPI از مدل SPMD [۷] هستند که در این مدل یک برنامه توسط کلیه پروسه‌ها اجرا می‌شود ولی پروسه‌ها بر روی بخش‌های مختلفی از داده‌ها کار می‌کنند. در انتها پروسه‌هایی که بصورت موازی در حال اجرا هستند با ارتباط با همدیگر نتیجه ی کلی عمل پردازش را تعیین می نمایند.

نصب و راه‌اندازی MPI.Net

برای نوشتن برنامه‌های موازی به کمک MPI.Net به چندین ابزار نیاز است. ذکر این نکته ضروری است که جهت نوشتن برنامه‌های MPI نیاز به داشتن کلاستر ویندوز و یا حتی ایستگاه‌کاری چندپردازنده/چند هسته نیست. بلکه فقط با داشتن یک ماشین که بر روی آن ویندوز XP یا بالاتر نصب باشد، می‌توان برنامه‌ی MPI نوشت.

- ✓ ویژوال استودیو ۲۰۰۵ و یا نسخه‌ی جدیدتر آن
- ✓ MPI میکروسافت: چندین روش جهت اجرای MS-MPI وجود دارد که انجام یکی از آنها کفایت می‌کند:
  - Microsoft Compute Cluster Pack SDK
  - Microsoft HPC Server 2008 or Microsoft Compute Cluster Server 2003
  - Windows Installer

ابتدا MPI.Net SDK [۸] را نصب نمایید. بعد از نصب، اولین برنامه‌ی MPI را اجرا نمایید. برای این منظور به محل نصب برنامه رفته و از داخل اعلان سیستم عامل برنامه‌ی نمونه‌ی pingpong.exe را اجرا نمایید. به عنوان خروجی عبارت زیر را مشاهده خواهید نمود:

```
C:\Program Files\MPI.NET>PingPong.exe
Rank 0 is alive and running on Nik-PCz
```

بنابراین برنامه‌ی pingpong.exe را با یک پروسه اجرا نموده‌اید. چون فقط یک پروسه داریم به آن rank صفر نسبت داده شده است و بر روی سیستمی با نام Nik-PC اجرا شده است.

تذکر: هنگام اجرای برنامه ممکن است با پیغام خطایی مشابه زیر مواجه شوید(از طرف دیواره‌ی آتش سیستم‌عامل) که برای اجرای برنامه فقط کافی است گزینه‌ی unblock را انتخاب نمایید.

می‌توان تعداد پروسه ها را که در اجرای برنامه‌ی ساده‌ی pingpong.exe شرکت دارند به تعداد مشخصی افزایش داد. هر کدام از این پروسه‌ها دارای شماره‌ی rank متفاوتی هستند. جهت اجرای چندین پروسه می‌توان از متد mpiexec که توسط SDK در اختیار شما قرار گرفته است، استفاده نمود.

```
C:\Program Files\MPI.NET>"C:\Program Files\Microsoft Compute Cluster
Pack\Bin\mpiexec.exe" -n 4 PingPong.exe
Rank 0 is alive and running on Nik-PC
Pinging process with rank 1... Pong!
  Rank 1 is alive and running on Nik-PC
Pinging process with rank 2... Pong!
  Rank 2 is alive and running on Nik-PC
Pinging process with rank 3... Pong!
  Rank 3 is alive and running on Nik-PC
```

همانطور که مشاهده می‌نماید برنامه‌ی mpiexec چهار پروسه‌ی متفاوت را به کار می‌گیرد که با همدیگر به عنوان یک برنامه‌ی واحد MPI کار می‌کنند.

تذکر: جهت نصب MPI.Net بر روی کلاستر باید فایل MPI.Net Run Time (از همان آدرس قبلی) را دانلود نموده و بر روی کلیه‌ی گره‌های کلاستر نصب نمایید.

تذکر مهم: اگر پیغام خطایی مبتی بر مسدود بودن پورت‌هایی به شما گزارش شد می‌توانید از درون دیواره‌ی آتش این پورت‌ها را باز نموده و یا اینکه دیواره‌ی آتش را در حین اجرای برنامه غیر فعال نمایید.!!!

نوشتن و اجرای یک برنامه‌ی ساده

به منظور نوشتن یک برنامه‌ی ساده‌ی MPI تست آن، یک پروژه‌ی جدید ویژوال استودیو (از نوع کنسولی) باز نمایید. سپس از داخل دایرکتوری LIB درون MPI.NET فایل MPI.dll را به پروژه Add Reference نمایید.

قبل از نوشتن برنامه فضاهای نام MPI و MPI_1 را به برنامه using نمایید. سپس باید مرحله‌ی مهمی را که در کلیه‌ی برنامه‌های MPI رعایت گردد، را به انجام برسانید.

این مرحله‌ی مهم مقداردهی محیط MPI است. کلیه‌ی پروسه های MPI قبل از اینکه بخواهند به هر طریقی از MPI بهره ببرند باید این مرحله‌ی حساس را انجام دهند. برای این منظور ابتدا یک نمونه از MPI.Environment را درون Main ایجاد می‌نماییم. بعد از آن این نمونه را بصورت ارجاعی به آرگومان‌های سطر فرمان برنامه پاس می‌دهیم.

```
using System;
using MPI;

class MPIHello
{
   static void Main(string[] args)
   {
      using (new MPI.Environment(ref args))
      {
         // MPI program goes here!
      }
   }
}
```

کل برنامه‌ی MPI باید درون بلاک using قرار گیرد، زیرا این بلاک تضمین می‌نماید که قبل از پایان یافتن برنامه، فضای محیط آزاد می‌گردد. کلیه‌ی برنامه‌های واقعی MPI باید دستورات مقداردهی و آزادسازی محیط را انجام دهند. همچنین یک ارجاع به آرگومان سطرفرمان args پاس داده می‌شود. زیرا پیاده‌سازی‌های MPI مجوز دارند به منظور ارسال اطلاعات Stateها به داخل روتین‌هایی که MPI را مقداردهی می‌نمایند، از پارامترهای ویژه‌ی سطر فرمان استفاده نمایند.

بنابراین با داشتن دستور using فوق، محیط MPI مقداردهی شده است و قادر خواهید بود تا برنامه‌ای ساده جهت چاپ یک عبارت توسط هر کدام از پروسه‌های در حال اجرا بنویسید. برای این منظور دستور زیر را به داخل عبارت using اضافه می‌نماییم.

```
Console.WriteLine("Hello, World! from rank " + Communicator.world.Rank
        + " (running on " + MPI.Environment.ProcessorName + ")");
```

هر کدام از پروسه‌های در حال اجرا این برنامه را به صورت کاملاً مستقل اجرا نموده و به طبع هر کدام خروجی مخصوص به خود را خواهند داشت. جهت اجرای برنامه، ابتدا فایل اجرایی آن را تولید نموده(MPIHello.exe)، بعد از آن برنامه را به صورت زیر اجرا نمایید.

```
C:\MPIHello\bin\Debug>mpiexec -n 8 MPIHello.exe
Hello, World! from rank 0 (running on NIK-PC)
Hello, World! from rank 6 (running on NIK-PC)
Hello, World! from rank 3 (running on NIK-PC)
Hello, World! from rank 7 (running on NIK-PC)
Hello, World! from rank 4 (running on NIK-PC)
Hello, World! from rank 1 (running on NIK-PC)
Hello, World! from rank 2 (running on NIK-PC)
Hello, World! from rank 5 (running on NIK-PC)
```

همانگونه که مشاهده می‌گردد، هشت خط خروجی متفاوت داریم که که هر کدام متناظر با یکی از پروسه‌هایی است که به عنوان جزیی از برنامه‌ی MPI در حال اجرا می‌باشند.

ارتباط دهنده‌ی[9] MPI (MPI Communicator)

همانگونه که در مثال قبل ملاحظه شد، برنامه‌ی داده شده فقط جهت یافتن rank هر کدام از پروسه‌ها به MPI Communicator مراجعه می‌کرد. MPI Communicator یک تجرید اساسی است که امکان ایجاد ارتباط بین پروسه‌های مختلف را می‌دهد و هر برنامه‌ی واقعی نیازمند به استفاده‌ی از آن را می‌باشد.

هر Communicator یک فضای ارتباطی برای مجموعه‌ای از پروسه‌ها بوجود می‌آورد. هر کدام از پروسه‌های درون این فضا قادر خواهند بود تا با بقیه‌ی پروسه‌ها ارتباط برقرار نمایند بدون اینکه نگران ایجاد تداخل و تصادم بین این پیام‌ها و پیام‌های ارسالی درون یک Communicator دیگر باشند. برنامه‌های MPI معمولاً از Communicatorهای متفاوتی جهت انجام کارهای مختلف بهره می برند. به عنوان نمونه ممکن است که برنامه‌ی Main از یک Communicator جهت کنترل پیام‌های کنترلی که توسط کاربر صادر می‌شوند استفاده نماید. در حالیکه یک مجموعه‌ی مشخصی از پروسه‌ها ممکن است از Communicator معینی جهت کار بر روی بخش ویژه‌ای از یک کار استفاده نمایند. از آنجا که هر کدام فضای ارتباطی مختص به خود را دارند نباید نگران ایجاد تداخل در پیام‌های کاربران و پیام‌هایی که پروسه‌ها جهت انجام کار با همدیگر ردوبدل می‌کنند، بود.

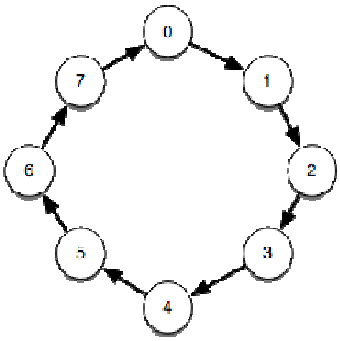
دو تا از خاصیت‌های اصلی Communicatorها که توسط برنامه‌ها استفاده می‌شوند عبارتند از rank و size که اولی شماره‌ی منحصربه‌فردی است که به هر کدام از پروسه‌ها داده می‌شود و دومی تعداد پروسه‌های در حال اجرا می‌باشد.

همچنین هر برنامه‌ی MPI دارای دو Communicator به نام‌های world و self می‌باشد. کلیه‌ی پروسه‌های در حال اجرای برنامه از طریق word با همدیگر به تبادل اطلاعات می‌پردازند. علاوه بر آن هر پروسه نیز یک Communicator مختص به خود را داراست که اطلاعات بسیار ناچیزی از آن پروسه را شامل می‌شود.

نحوه‌ی تبادل پیام بین پروسه‌ها

ارتباطات نقطه به نقطه ابتدایی‌ترین شکل ارتباط در MPI است که به برنامه اجازه می دهد تا یک پیام را از یک پروسه به پروسه‌ی دیگری ارسال نماید. هر پیام دارای پروسه‌ی مبداء و مقصد می‌باشد(به کمک rank هر پروسه آنها را از همدیگر تفکیک می نماید) و یک برچسب داخلی که نوع پیام را مشخص می‌نماید و بخش داده‌ای که باید ارسال گردد. در کل دو نوع ارتباط جهت ارسال و دریافت پیام نقطه به نقطه در MPI.Net وجود دارد که عبارتند از روش blocking و روش non-blocking .(یکی با اعمال قفل و دیگری بدون عمل قفل گذاری). به عنوان نمونه یک پیام ارسالی از نوع قفل شدنی تا زمانی که پیام به درون بافر داخلی MPI (جهت ارسال) کپی نشود، بلوکه می‌گردد. همچنین یک پیام دریافت از نوع قفل شدنی نیز تا هنگامی که پیام به طور کامل دریافت شده و بازگشایی گردد، منتظر خواهد ماند. در حالیکه مدل بدون قفل، یک ارتباط را مقداردهی می‌نماید بدون اینکه منتظر تکمیل ارتباط باقی بماند.

مثال اولی از مدل ارتباطی نقطه به نقطه می‌باشد به اینصورت که برنامه‌ای نوشته خواهد شد که یک پیام را دور یک حلقه ارسال می‌کند. روش کار به اینصورت است که ارسال پیام توسط یکی از پروسه‌ها شروع می‌شود، سپس از یک پروسه به پروسه‌ی بعد منتقل می‌گردد تا در انتها به پروسه‌ی تولید کننده‌ی پیام می‌رسد. شکل نحوه‌ی پاس دادن پیام توسط پروسه‌ها را در زیر می‌بینید.



در زیر شمایی کلی از برنامه را می بینید.

```
using System;
```

```
using MPI;
class Ring
{
   static void Main(string[] args)
   {
      using (new MPI.Environment(ref  args))
      {
         Intraommunicator comm = Communicator.world;
         if (comm.Rank == 0)
         {
            // program for rank 0
         }
         else // not rank 0
         {
            // program for all other ranks
         }
      }
   }
}
```

همانگونه که در الگوی فوق مشاهده می‌گردد یکی از پروسه‌ها (که معمولاً ریشه نامیده می‌شود و دارای rank صفر می‌باشد) معمولاً دارای کدی متفاوت از بقیه پروسه‌ها می‌باشد و معمولاً اعمال هماهنگی و ارتباط با مشتری به عهده‌ی این پروسه می‌باشد.

پروسه‌ی rank0 معمولاً مسول ایجاد ارتباط می‌باشد (با ارسال پیام به rank1). کد زیر نحوه‌ی ارسال مقداری داده را به روش قفل‌شدنی نشان می‌دهد. سه پارامتر متد Send به ترتیب عبارتند از:

- اطلاعاتی که باید ارسال شوند.
- Rank پروسه‌ی مقصد که در این مثال ابتدا پیام به rank1 ارسال می‌گردد.
- برچسب(tag) پیام. از این برچسب گیرنده جهت تشخیص این پیام از بقیه استفاده می‌کند. اگر فقط یک نوع پیام داشته باشیم از برچسب با مقدار صفر استفاده خواهیم نمود.

```
if (comm.Rank == 0)
{
   // program for rank 0
   comm.Send("Nik", 1, 0);

   // receive the final message
}
```

حال باید کد بقیه پروسه ها را نوشت. پروسه‌ها منتظر می‌مانند تا پیامی از پروسه‌ی قبلی دریافت نمایند با دریافت پیام آنرا در خروجی چاپ نموده و به پروسه ی بعدی ارسال می‌نمایند.

```
else // not rank 0
{
   // program for all other ranks
   stringmsg = comm.Receive<string>(comm.Rank - 1, 0);

   Console.WriteLine("Rank " + comm.Rank + " received message \"" + msg + "\".");

   comm.Send(msg + ", " + comm.Rank, (comm.Rank + 1) % comm.Size, 0);
}
```

همانگونه که در کد فوق مشاهده می نمایید، هر کدام از پروسه‌ها پیام را از پروسه‌ی قبلی خود(comm..Rank-1) دریافت می‌نمایند و بعد از چاپ آن، پیام به پروسه ی بعدی ارسال می‌گردد. در انتها به حالت خاصی می‌رسیم که در آن پروسه‌ی اول اطلاعات را از پروسه‌ی آخر دریافت می نماید. کد اصلاح شده‌ی پروسه‌ی اول به فرم زیر خواهد بود.

```
if (comm.Rank == 0)
{
   // program for rank 0
   comm.Send("Rosie", 1, 0);

   // receive the final message
   stringmsg = comm.Receive<string>(Communicator.anySource, 0);

   Console.WriteLine("Rank " + comm.Rank + " received message \"" + msg + "\".");
}
```

با توجه به اینکه تعداد پروسه‌ها نامعین می باشد در قسمت آدرس پروسه‌ی قبلی مقدار Communicator.AnySource قرار داده شده است. با اجرای برنامه خروجی زیر را خواهیم داشت:

```
C:\Ring\bin\Debug>mpiexec -n 8 Ring.exe
Rank 1 received message "Nik".
Rank 2 received message " Nik, 1".
Rank 3 received message " Nik, 1, 2".
Rank 4 received message " Nik, 1, 2, 3".
Rank 5 received message " Nik, 1, 2, 3, 4".
Rank 6 received message " Nik, 1, 2, 3, 4, 5".
Rank 7 received message " Nik, 1, 2, 3, 4, 5, 6".
Rank 0 received message " Nik, 1, 2, 3, 4, 5, 6, 7".
```

در این مثال فقط یک رشته‌ی ساده بین پروسه‌ها ردوبدل شده است اما در حالت کلی انواعی که می‌توانند بین پروسه‌ها جابجا شوند عبارتند از:

- انواع پایه زبان (primitive Type)
- ساختارهای عمومی تعریف شده توسط کاربر (public struct)
- کلیه‌ی انواع ایجاد شده توسط کلاس‌های قابل serialize(Serializable Class)

ارتباطات گروهی(Collective Communication)

در ارتباطات گروهی، کلیه‌ی پروسه‌های دورن مجموعه جهت انجام یک عمل با همدیگر همکار می کنند. در برنامه‌های MPI معمولاً هر کدام از پروسه‌ها بدون توجه به کاری که بقیه انجام می‌دهند، مشغول انجام عملیات خودشان می‌باشند به جز در مواقعی که منتظر یک ارتباط بین‌پروسه‌ای هستند. در اغلب برنامه‌های موازی پروسه‌ها بصورت تقریباً مستقل مشغول به کار هستند. با این وجود بعضی از مواقع لازم است که اطمینان حاصل نماییم که پروسه‌ها در یک لحظه مشغول انجام کار یکسانی هستند.

متد **Barrier** برای این منظور به کار گرفته می‌شود. هنگامی که یک پروسه وارد barrier می‌شود از آن خارج نمی‌گردد تا کلیه پروسه‌ها به Barrier وارد شوند. به عنوان نمونه در مثال زیر هر کدام از مراحل حلقه کاملاً همگام هستند. بگونه‌ای که همه‌ی پروسه‌ها در یک زمان خاص در تکرار یکسانی از حلقه می‌باشند.(تکرار یکسانی از حلقه را اجرا می‌نمایند)

```
using System;
using MPI;

class Barrier
{
   static void Main(string[] args)
   {
      using (new MPI.Environment(ref args))
      {
         Intracommunicator comm = Communicator.world;
         for (inti = 1; i<= 5; ++i)
         {
            comm.Barrier();
            if (comm.Rank == 0)
               Console.WriteLine("Everyone is on step " + i + ".");
         }
      }
   }
}
```

اجرای برنامه با هر تعداد پروسه نشان می‌دهد که همه در تکرار یکسانی از حلقه هستند. به عنوان نمونه در مثال زیر هشت پروسه در حال اجرا هستند.

```
C:\Barrier\bin\Debug>mpiexec -n 8 Barrier.exe
Everyone is on step 1.
Everyone is on step 2.
Everyone is on step 3.
Everyone is on step 4.
Everyone is on step 5.
```

متد مهم دیگری که در MPI وجود دارد متد **Gather** است که اطلاعات کلیه‌ی پروسه‌ها را که در مجموعه در حال اجرا هستند بر روی یک پروسه جمع‌آوری می‌نماید. به عنوان نمونه در مثال زیر اسامی کلیه‌ی ماشین‌هایی که پروسه‌ها بر روی آنها در حال اجرا هستند جمع‌آوری شده و توسط پروسه‌ی ریشه در خروجی چاپ می‌شوند.

```
using System;
using MPI;
class Hostnames
{
   static void Main(string[] args)
   {
      using (new MPI.Environment(ref args))
      {
         Intracommunicator comm = Communicator.world;
         string[] hostnames = comm.Gather(MPI.Environment.ProcessorName, 0);
         if (comm.Rank == 0)
         {
            Array.Sort(hostnames);
            foreach(string host in hostnames)
               Console.WriteLine(host);
         }
      }
   }
}
```

حالت دیگری که توسط دو متد broadcast و scatter پیاده‌سازی می‌گردد، عکس متد gather است. در این متدها می‌توان اطلاعات را بین پروسه‌ها پخش نمود. متد broadcast مقداری را از یک پروسه گرفته و آنرا بین بقیه‌ی پروسه‌ها پخش می‌نماید. به عنوان نمونه مثالی را درنظر بگیرید که در آن یک پروسه مقداری را از کاربر گرفته و آن را بین بقیه‌ی پروسه‌ها که بصورت موازی در حال اجرا هستند پخش می‌نماید. مثال زیر بیانگر چگونگی انجام عمل فوق می‌باشد.

```
using System;
using MPI;
```

```
class CommandServer
{
   static void Main(string[] args)
   {
      using (new MPI.Environment(ref args))
      {
         Intracommunicator comm = Communicator.world;
         string command = null;
         do
         {
            if (comm.Rank == 0)
               command =Console.ReadLine();

            // distribute the command
            comm.Broadcast(ref command, 0);

            // each process handles the command
         } while (command != "quit");
      }
   }
}
```

آرگومان‌های متد broadcast عبارتند از مقداری که باید پخش شود و شماره‌ی پروسه‌ای که این مقدار را جهت پخش آماده نموده است.

تذکر: scatter مشابه broadcast است با این تفاوت که می توان مقادیر متفاوتی را به پروسه‌های مختلف ارسال نمود.

متد AlltoAll داده‌ها را از هر پروسه‌ای به بقیه ارسال می‌نماید. هر پروسه دارای آرایه‌ای است که iام المان به پروسه‌ی با Rank i ارسال خواهد شد. همچنین هر پروسه بعد از ارسال، پروسه‌هایی دریافت خواهد کرد که iامین مقدار آرایه مقداری است که از پروسه‌ی iام رسیده است. حال مثال زیر را درنظر بگیرید که در آن آرایه‌ای به اندازه‌ی تعداد پروسه‌ها داریم و این آرایه از طرف هر کدام از پروسه‌ها به همه ارسال می‌شود.

```
string[] data = new string[comm.Size];
for (intdest = 0; dest<comm.Size; ++dest)
   data[dest] = "From " + comm.Rank + " to " + dest;

string[] results = comm.Alltoall(data[]);
```

اگر برنامه مشابه حالت‌های قبلی با هشت پروسه اجرا شود، خروجی‌های زیر را در پروسه‌ی یک (Rank 1) مشاهده می‌شود.

```
From 0 to 1.
From 1 to 1.
From 2 to 1.
```

```
From 3 to 1.
From 4 to 1.
From 5 to 1.
From 6 to 1.
From 7 to 1.
```

ترکیب نتایج با موازی سازی

MPI دارای تعدادی متد جهت انجام موازی‌سازی است. این متدها مقادیری را که توسط هر کدام از پروسه‌ها ارئه می‌شوند با همدیگر ترکیب کرده و نتیجه‌ی کلی را محاسبه می‌نمایند. روشی که توسط آن نتایج با همدیگر ترکیب می‌شوند معمولاً توسط کاربر تعیین می‌گردد. به عنوان نمونه کاربر می‌تواند عمل جمع، ضرب و یا ماکزیمم‌گیری را انجام دهد.

یکی از جمع‌آوری‌کننده‌ها reduce نام دارد که نتایج ده دست آمده از پروسه‌ها را با همدیگر ترکیب نموده و نتیجه را به پروسه‌ی ریشه تحویل می‌دهد. اگر پروسه‌ی rank i نتیجه‌ی vi را بدهد نتیجه ی reduction روی روی n پروسه عبارت خواهد بود از v1+v2+v3+...+vn که عمل + می‌تواند هر عمگر دیگری نیز باشد.

جهت روشن‌تر شدن مطلب مثالی از نحوه‌ی تخمین عدد pi را ارائه می‌گردد. الگوریتم ساده به این شکل است که دایره‌ی واحدی را درون مربع واحدی رسم می‌نماییم. سپس خطوطی (فلش) را درون دایره می‌کشیم. تعداد خطوطی که درون دایره واقع می شوند به تعداد خطوطی که درون مربع واقع می‌شوند برابر است با نسبت مساحت دایره به مساحت مربع. ازاین ایده‌ی ساده می توان جهت تخمین عدد pi (چهار برابر نسبت مساحت دایره به مساحت مربع) استفاده کرد. ابتدا الگوریتم خطی آنرا می بینیم.

```
using System;
class SequentialPi
{
   static void Main(string[] args)
   {
      int dartsPerProcessor = 10000;
      Random random = new Random();
      int dartsInCircle = 0;
      for (inti = 0; i<dartsPerProcessor; ++i)
      {
         double x = (random.NextDouble() - 0.5) * 2;
         double y = (random.NextDouble() - 0.5) * 2;
         if (x * x + y * y <= 1.0)
            ++dartsInCircle;
      }

      Console.WriteLine("Pi is approximately {0:F15}.",
```

```
                4*(double)totalDartsInCircle/(double)dartsPerProcessor);
    }
}
```

جهت انجام موازی‌سازی سازی می‌توان به این شیوه عمل کرد که چندین پروسه داشته باشیم که بصورت مستقل هر کدام شروع به رسم dart نمایند. بعد از اتمام ترسیم می‌توان نتایج را جمع‌آوری نموده و مقدار pi را تخمین زد.

جهت محاسبه‌ی تعداد نقاطی که درون دایره واقع شده‌اند می‌توان از متد Reduce به فرم زیر استفاده نمود.

```
int totalDartsInCircle = comm.Reduce(dartsInCircle, Operation<int>.Add, 0);
```

همانگونه که مشاهده می‌گردد این متد یکی متغیری را دریافت می‌نماید که باید مقادیر آنرا از پروسه‌های متفاوت جمع‌آوری نماید و به عنوان پارامتر دوم نیز نوع عملی که قرار است بر روی این مقادیر اعمال گردد که در اینجا فراخوانی متد Add است. پارامتر آخر هم rank پروسه‌ی ریشه می‌باشد.

کد کامل شده‌ی برنامه را در زیر می‌بینید:

```
using System;
using MPI;
class Pi
{
   static void Add( int x ,int y) {return x+y;}

   static void Main(string[] args)
   {
      using (new MPI.Environment(ref args))
      {
         Intracommunicator comm = Communicator.world;
         int dartsPerProcessor = 10000;
         Random random = new Random(5 * world.Rank);
         int dartsInCircle = 0;
         for (inti = 0; i<dartsPerProcessor; ++i)
         {
            double x = (random.NextDouble() - 0.5) * 2;
            double y = (random.NextDouble() - 0.5) * 2;
            if (x * x + y * y <= 1.0)
               ++dartsInCircle;
         }

         int totalDartsInCircle = comm.Reduce(dartsInCircle, Operation<int>.Add, 0);
         if (comm.Rank == 0)
            Console.WriteLine("Pi is approximately {0:F15}.",
                  4*(double)totalDartsInCircle/(world.Size*(double)dartsPerProcessor));
```

```
    }
  }
}
```

در زیر دو نمونه از اجرای برنامه دیده می‌شود.(تعداد اعداد تصادفی هر پروسه برابر ۱۰۰۰۰۰۰۰ قرار داده شده است.)

```
C:\Program Files\Microsoft Compute Cluster Pack\Bin>mpiexec -n 40 mpi-1
Pi is approximately 3.1414554800000000.

C:\Program Files\Microsoft Compute Cluster Pack\Bin>mpiexec -n 100 mpi-1
Pi is approximately 3.1415300120000000.
```

منابع و مراجع:

1- http://osl.iu.edu/research/mpi.net/
2- http://resourcekit.windowshpc.net/MORE_INFO/MPI.NET_Tutorial.html
3- سایت مایکروسافت
4- http://www.codeproject.com/KB/WCF/mandelbrotapi.aspx

[1] یک کتابخانه بر روی مجموعه .Net که امکان ایجاد برنامه‌های کاربردی با کارایی بالا(High Performance) و موازی را می دهد.
[2] Communicator
[3] Point to Point Communication
[4] Collective
[5] Multi-thread Workstation
[6] Program State
[7] Single program Multiple Data
[8] http://osl.iu.edu/research/mpi.net/files/1.0.0/MPI.NET SDK.msi
[9] Communicator

توجه: فایل‌های موردنیاز جهت نصب و اجرای برنامه‌های MPI بر روی سایت http://webpages.iust.ac.ir/balouchzahi/courses/mpi.htm آپلود شده است.