혼합격자를 이용한 2차원 난류 유동장 해석 프로그램의 병렬화

Parallelization of a Two-Dimensional Navier-Stokes Solver Using Hybrid Meshes

○옥호남[1], 박승오[2]
Honam Ok and Seung-O Park

A two-dimensional Navier-Stokes solver using hybrid meshes is parallelized with a domain decompostion method. The focus of this paper is placed on minimizing the amount of effort in parallelizing the serial version of the solver, and this is achieved by adding an additional layer of cells to each decomposed domain. Most subroutines of the serial solver are used without modification, and the information exchange between neighboring domains is achieved using MPI(Message Passing Interface) library. Load balancing among the processors and scheduling of the message passing are implemented to reduce the overhead of parallelization, and the speed-up achieved by parallelization is measured on the transonic invisicd and turbulent flow problems. The parallelization efficiencies of the explicit Runge-Kutta scheme and the implicit point-SGS scheme are compared and the effects of various factors on the results are also studied.

1. 서론

복잡한 형상 주위 난류 유동장의 효율적 해석을 위해 도입된 혼합격자 기법은 정렬격자를 이용하는 해석 방법에 비하여 아직까지 큰 기억용량과 많은 양의 계산 시간을 필요로 한다. 따라서 개발된 프로그램을 설계도구로서 유용하게 사용하기 위해서는 고속 대용량의 super-computer의 활용이 필요하며, 또한 이 프로그램이 super-computer에서 높은 계산 효율을 얻을 수 있도록 만들어져 있어야 한다. 최근의 super-computer 개발 동향을 살펴보면, Cray C90과 같은 vector register를 통한 pipeline 개념을 이용한 vector형 super-computer로부터 저가의 workstation용 CPU를 고속 bus로 연결한 병렬형 super-computer로 점차 전환되어 가는 양상을 보이고 있다.

그러나 병렬형 super-computer는 vector형 super-computer에 비하여 계산효율의 증대를 위한 대폭적인 프로그램 수정을 요구하는 경우가 많으며, 또한 이 작업 역시 해석 프로그램 자체의 개발에 버금가는 많은 노력을 필요로 한다. 프로그램의 병렬화를 위해서는 계산영역을 계산에 사용될 processor 수만큼의 작은 영역으로 분할하고 이웃 processor와 주고받아야 할 정보는 별도의 library를 이용하는 영역분할법(domain decomposition method)이 주로 사용되고 있으며, 이를 이용하기 위해서는 영역분할 및 정보 전달에 관련된 부분에 대한 추가적인 프로그램 작업이 필요하다.

본 연구에서는 개발된 혼합격자를 이용한 2차원 난류 유동장 해석 프로그램[1,2]의 병렬화를 수행하였으며, 특히 병렬화를 위한 프로그램 수정 요구를 최소화할 수 있도록 하는 데에 초점을 맞추었다. 이를 위해 분할된 각각의 영역에 추가적인 fringe cell을 도입하는 방법을 채택하였으며, 그럼으로써 단일 processor용으로 개발된 프로그램의 주요 계산 module들이 거의 수정 없이 사용될 수 있게 하였다. Processor간의 정보 전달에는 여러 종류의 기종에서 사용이 가능하고 성능도 우수한 MPI(Message Passing Interface)[3]를 사용하였으며, 실제 프로그램의 병렬화에는 100여개 이상의 MPI function중 몇 개만이 필요하였다. 양해법(explicit scheme)인 4 stage Runge-Kutta 기법 및 음해법(implicit scheme)인 point-SGS(Symmetric Gauss-Seidel) 기법의 병렬 효율성을 NACA0012 익형 주위의 천음속 비점성 및 난류 유동에 대하여 산출하였으며, 계산에 사용된 processor의 특성 및 영역분할 기법 등이 결과에 미치는 영향을 분석하였다.

2. 수치해석 방법

복잡한 형상에 대한 효율적 해석을 위해 개발된 혼합격자 해석 프로그램인 UNS2D(Unstructured Navier-Stokes solver in 2 Dimensions)에 대한 자세한 설명은 참고문헌 [1,2]에 자세히 기술되어 있으며, 여기서는 개요만을 간략히 나타내고자 한다.

1) 한국항공우주연구소 공력성능연구그룹 (305-333 대전시 유성구 어은동 52번지 Tel : 042-860-2317)
2) 한국과학기술원 항공우주공학과 (305-701 대전시 유성구 구성동 373-1 Tel : 042-869-3713)

격자는 어떤 형태의 다각형도 처리할 수 있도록 되어 있으나, 경계층과 같이 한쪽 방향으로의 격자 밀집이 필요한 부분은 사각형으로, 그 외의 영역은 삼각형으로 채우는 혼합격자를 주로 사용하며, 유동방정식은 유동변수의 값이 격자의 중심에서 정의되는 격자중심 유한체적법(cell-centered finite volume method)으로 적분하였다. 비점성항의 차분화에는 Roe의 FDS(Flux Difference Splitting) 기법[4]을 사용하였으며, 2차의 정확도를 얻으면서도 해의 불연속을 포착할 수 있도록 min-mod type [5] 혹은 Venkatakrishnan[6]의 제한자(limiter)를 사용하였다. 점성항의 차분화도 제어면(face)에서 필요로하는 유동변수의 기울기를 적절한 제어체적에 Green 공식을 적용하여 구함으로써 2차의 정확도를 갖도록 하였다. 시간 적분은 양해법인(explicit scheme)인 multi-stage Runge-Kutta scheme과 음해법(implicit scheme)인 point-SGS(Symmetric Gauss Seidel) scheme[7]을 사용하였으며, 정상해로의 수렴을 돕기 위해 implicit residual smoothing 기법과 local time stepping 기법을 사용하였다. 난류 유동장의 해석을 위해서는 Spalart-Allmaras의 one-equation model[8]을 도입하였으며, 유동방정식과 결합하여 풀어주었다. 개발된 프로그램의 정확도 및 효율성은 다양한 아음속 및 천음속 난류 유동장 해석을 통하여 검증되었으며, 자세한 사항은 참고문헌 [1,2]에 기술되어 있다.

3. 병렬처리 기법

프로그램의 병렬화는 계산영역을 계산에 사용될 processor 수만큼의 작은 영역으로 분할하여 각각의 processor가 계산을 담당하게 하는 영역분할법(domain decomposition method)을 사용하였다. 영역을 분할함에 있어서 각 processor가 담당해야 할 계산 양이 적절한 균형을 이루도록 하여야 하며, 또한 병렬형 super-computer에서 가장 큰 overhead로 작용하는 각 processor간 정보 전달에 소요되는 시간을 최소화할 수 있어야 할 것이다. 그러나 무엇보다도 본 연구에서는 개발된 프로그램이 single CPU workstation, NOW(Network Of Workstations), PVP(Parallel Vecotor Processors), dedicated parallel machine 등 다양한 기종에서 수행이 가능하도록 하는데 역점을 두었으며, 가능하면 대부분의 subroutine이 수정 없이 모든 기종에서 그대로 사용될 수 있도록 하였다.

3.1 Code 병렬화 기법

영역분할기법을 이용한 프로그램의 병렬화에 있어서 가장 주의를 요하는 부분이 영역분할에 의해 생긴 내부 경계면(internal interface)을 어떻게 처리하는가 하는 것이며, 이에 따라 계산 효율 및 프로그램의 수정 정도가 결정되는 중요성을 지니고 있다. 즉, 내부 경계면에서의 정확한(serial version과 정확도가 같은) flux 혹은 미지수의 값을 얻기 위해서는 적절한 계산 단계에서 이웃 processor로 부터 유동장의 정보를 가져와야만 하며, 이를 위해서는 내부 경계면을 공유한 processor간의 정보 전달(information exchange)이 수반되어야만 한다. 이를 위해서는 일반적으로 다음 두 가지의 방법을 생각해 볼 수 있을 것이다.[9]

첫 번째 방법은 필요로 하는 물리량(미지수, flux, gradient 등)에 대한 정보를 각각의 processor가 모두 다 계산하여 가지고 있으나, 정보의 부족으로 계산이 불가능한 항이나 추가되어야 할 부분은 이웃 processor에서 가져온다. 다음으로 모든 flux의 계산이 끝나면 이를 이용하여 각각의 processor에서 미지수를 계산해 주면 된다. 이 방법은 각 processor에서 꼭 필요한 정보만을 저장하며, 또한 꼭 필요한 계산만을 수행하기 때문에 기억용량 및 계산량의 추가적인 증가는 없다. 그러나 이 경우에는 각 계산 단계에서 이웃 processor에서 얻을 정보를 항상 고려해 두어야 하므로 프로그램의 대폭적인 수정이 필요로 하며, 또한 정보전달 양이 증가하여 이에 소요된 overhead가 CPU 시간 절감분을 초과 함으로써 실제 계산시간이 다음에 설명할 방법보다 오히려 증가할 가능성도 있다.

다음 방법은 영역분할에 의해 생성된 각각의 계산 영역에 한 겹의 fringe layer를 추가하는 것이다. 그림 1에서 계산 영역을 4개의 영역으로 나누었으나, 각 processor에서 실제로 가져가는 계산 영역은 내부가 칠해진 영역이 추가된 것이다. 이 부분은 이웃 processor에서 정확한 계산이 가능한 영역이며, 각 processor 배부에서는 정확한 계산이 이루어 질

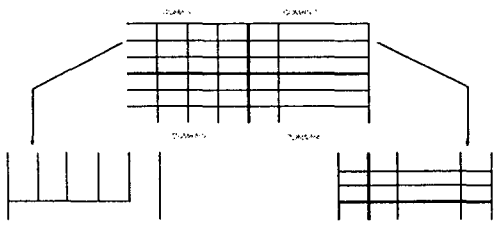

Fig. 1 Fringe layer 도입을 통한 프로그램 병렬화

수 없는 부분이다. 그러나 이 정보를 포함하고 있음으로 해서 원래의 계산 영역(그림 1에서 굵은 선으로 표시된 경계면 내부의 격자)에 대해서는 정확한 계산이 이루어 질 수 있게 된다. 또한 정보의 전달도 각 processor에서의 계산이 모두 끝난 후 이 fringe layer에 있는 격자에서의 미지수 값을 이웃 processor로부터 가져오기만 하면 되므로, 앞의 방법에 비하여 상당히 줄어들게 되며, 따라서 fringe layer의 존재로 인하여 약간 증가된 계산시간을 충분히 만회할 수 있을 것이다. 그러나 가장 큰 이점은 이 방법을 사용하는 경우 각 processor에서는 이웃 processor로부터 얻을 정보와 무관하게 독자적인 계산을 수행하면 되므로 프로그램의 수정을 최소화할 수 있게 된다는 것이다. 따라서 본 연구에서는 이와 같은 fringe layer를 이용하는 방법을 선택하였으며, 이에 의해 serial version의 프로그램에서 사용된 subroutine의 거의 모두를 수정 없이 사용할 수 있게 되었다.

3.2 영역분할 및 부하 균등화 기법

혼합격자의 형성은 유동장의 변화가 심하거나 격자 형성의 효율성을 기하기 위해 필요한 영역에는 사각형 격자를 만들고 그 외의 영역은 삼각형으로 채우는 방법으로 이루어 졌다. 필요한 경우 사각형이나 삼각형 외의 다른 격자 모양을 취할 수도 있으나, 이러한 형태의 격자를 처음부터 만들기는 어려우며, 단지 격자의 적응을 통하여 다양한 모양의 다각형 격자를 만들 수는 있다. 일단 격자가 형성되면 계산에 사용하고자 하는 processor의 수만큼 계산영역을 작은 영역(sub-domain)으로 분할하여야 하며, 이를 위해 여러 가지 방법이 개발되어 있다. 본 연구에서는 항공기나 고속 비행체의 설계에 직접 사용될 수 있는 프로그램의 개발을 목표로 하고 있으며, 이는 대부분 복잡한 형상을 포함하고 있다. 이 경우 대부분 형상의 modeling을 위하여 CAD 프로그램을 사용하며, 이에 대한 격자형성 프로그램은 CAD 자료의 처리가 가능하여야 한다. 그러나 이러한 능력을 갖춘 격자 형성 프로그램의 작성은 그것만으로도 엄청난 노력과 투자를 필요로 하는 일이며, 따라서 본 연구에서는 격자형성 및 영역분할을 위해 상용 프로그램인 Fluent 5[10]의 pre-processor를 사용하였다.

격자의 분할은 프로그램의 성능이나 계산에 사용될 hardware의 특성에 따라 주안점을 두어야 할 요소가 달라질 수 있으나, 일반적으로 주의해야할 요소는 다음 세 가지로 요약될 수 있을 것이다.

- 각 분할영역에 포함된 격자수의 균형
- 영역분할에 의해 생긴 내부 경계면 크기의 최소화
- 분할영역에 접하는 이웃 영역 수의 최소화

첫 번째 요소는 각 processor에 걸리는 계산 부하의 균형을 맞추어 다른 processor가 계산을 수행하고 있을 때 쉬고 있는 processor가 없도록 하기 위한 것으로 부하균등화(load balancing) 기법이라 불린다. 각 processor간에 전달해야 하는 정보의 양은 fringe layer에 들어 있는 격자의 수에 정비례하며, 이는 결국 내부 경계면의 크기에 직접적인 영향을 받게 된다. 따라서 내부 경계면의 크기를 최소화함으로써 전달되어야만 하는 정보의 양을 줄일 수 있을 것이다. 마지막 요소 역시 정보 전달에 관련된 것으로, 이웃 영역의 수가 늘어날수록 여러 processor와의 정보 전달을 시도해야 한다. 병렬 컴퓨터에 있어서 정보 전달에 소요되는 시간은 크게 두 부분으로 나눌 수 있으며, 첫 번째는 각 processor간의 정보 전달을 초기화하는 과정이며 다음이 실제로 정보를 주고받는 것이다. 그런데 이 초기화에 소요되는 시간을 latency라 하며, 오히려 이에 소요되는 시간이 다음 단계인 실제 정보를 주고받는데 걸리는 시간 보다 긴 경우도 많다. 따라서 latency가 큰 hardware일수록 이웃 영역의 수를 최소화하는 것이 계산 시간의 단축에 필수적이다. Fluent 5의 pre-processor에는 이를 고려한 여러 가지 영역분할법이 포함되어 있으며, 여기서는 계산영역의 principal axis를 따라 차례대로 2등분씩 나누는 bisection method를 주로 사용하였다.

그림 2에는 slat 및 flap이 펼쳐진 3요소 익형 주위의 혼합격자를 나타내었으며, 4개의 영역으로 격자가 분할되어 있다. 표 1에는 각 분할영역의 격자에 대한 통계를 보여주고 있으며, I-Cell 및 I-Node는 내부 경계면과 접하거나 경계면에 놓인 격자와 격자점의 개수를 나타낸다. 영역분할에 의해 격자수가 거의 균등하게 4등분되어 있음을 알 수 있으며, 경계면 격자와 격자점의 수는 이에 비해 덜 균등하게 되어 있다. 본 연구에서는 fringe layer를

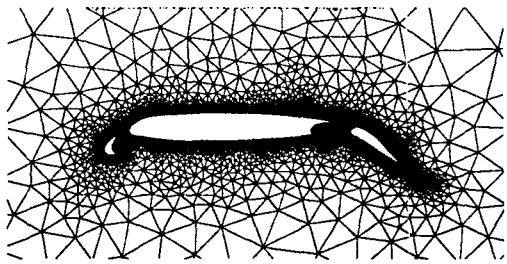

Fig. 2 4개 영역으로 분할된 혼합격자

Table 1 계산영역 4등분에 의한 분할영역 격자

Domain ID	Num. of Cells	Num. of I-Cells	Num. of Nodes	Num. of I-Nodes	Num. of Neighbors
1	3,184	58	1,818	61	3
2	3,188	93	2,205	98	3
3	3,190	110	2,164	20	3
4	3,195	78	2,083	89	3
All	12,757	N/A	8,084	N/A	N/A

원래의 분할영역에 추가하여 각 processor에 할당하게 되므로, 계산에 사용될 분할격자는 표 2와 같은 특성을 보인다. 전체적으로 fringe layer의 추가에 의하여 격자(cell) 및 격자점(node)의 수가 약 5.3% 증가하였으며, 경계면에 놓인 격자 및 격자점은 훨씬 더 큰 증가를 보인다. 이와 같은 격자의 특성에 의하면 processor의 성능이 격자의 크기에 무관하게 일정한 경우(Cray T3E의 경우 격자 크기가 작아짐에 따라 계산 성능이 크게 향상됨) 프로그램의 병렬화에 의해 최소한 7.0% 이상 계산량의 overhead가 발생함을 의미한다.

3.3 Scheduling 기법

각 processor는 내부 경계면의 존재 여부에 상관없이 serial version의 solver와 같은 subroutine을 사용하여 flux의 계산 및 격자중심에서의 해를 구한다. 이때 fringe layer에 놓인 격자에 대한 flux 및 해는 정확한 값을 갖지는 못하며, 따라서 이 단계가 끝나면 fringe layer에서의 해를 정확한 값으로 바꾸어 주어야 한다. 이때 임의의 processor의 fringe layer 격자에서의 해는 다른 processor들 중 최소한 하나는 정확한 계산 결과를 얻었을 것이며 이에 대한 정보를 주고받아야 한다. 그러나 필요한 정보를 동시에 주고받으려 하는 경우 심각한 정보 전달 병목현상(bottleneck)을 야기하며, 경우에 따라서는 정보 전달이 더 이상 진행되지 못하는 deadlock이 발생할 수 있다. 따라서 이러한 병목현상 및 deadlock을 피할 수 있도록 하여야 하며, 정보 전달에 따르는 overhead를 최소화하기 위해서는 동시에 여러 쌍의 processor간의 정보 전달이 이루어 질 수 있어야 한다. 본 연구에서는 Fluent 5의 pre-processor를 이용한 격자 형성 및 영역 분할이 끝나면 각각의 분할영역에 fringe layer를 추가하고, 다음으로 정보 전달에 필요한 list를 작성하였다. 이 list를 근거로 최대한의 많은 processor들이 동시에 정보 전달에 참여하면서도 bottleneck 및 deadlock을 피할 수 있도록 정보 전달의 단계를 설정하였다. 표 3에는 표 2에 나타낸 분할영역 격자들의 정보 전달 체계를 나타내고 있다. 화살표는 정보 전달의 방향을 나타내고 있으며, 괄호 안의 숫자는 정보를 전달이 필요한 격자의 수를 나타낸다. 4개의 processor가 각각 다른 processor와 정보를 주고받아야 하므로 총 12번의 정보 전달이 있어야 하며, 이를 독립적인 쌍으로 구성하면 모두 6단계의 정보 전달이 이루어져야 한다. 표 2에서 보면 processor 1에서 fringe layer에 포함

Table 2 Fringe layer가 추가된 4등분 분할영역 격자

Domain ID	N of Cells (% Inc.)	N of I-Cells (% Inc.)	N of Nodes (% Inc.)	N of I-Nodes (% Inc.)
1	3,301 (3.7)	117 (101.7)	1,887 (3.8)	69 (13.1)
2	3,378 (6.0)	190 (104.3)	2,318 (5.1)	113 (15.3)
3	3,413 (7.0)	223 (102.7)	2,296 (6.1)	132 (560.0)
4	3,343 (4.6)	148 (89.7)	2,170 (4.2)	87 (-2.2)
SUM	13,435 (5.3)	678 (100.0)	8,671 (5.3)	401 (49.6)

Table 3 4개 Processor간의 정보 전달 schedule

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6
1 → 2 (93)	1 → 3 (5)	3 → 2 (85)	2 → 1 (90)	3 → 1 (3)	4 → 1 (24)
2 → 4 (117)	2 → 4 (7)	1 → 4 (24)	4 → 3 (133)	4 → 2 (11)	3 → 2 (86)

된 격자의 수는 총 117개이며, 여기에 필요한 정보는 표 3을 보면 processor 2로부터 stage 4에서 90개, processor 3으로부터 stage 5에서 3개, 그리고 processor 4로부터 stage 6에서 24개를 받아들여 미지수를 모두 새로운 값으로 채울 수 있음을 알 수 있다. 또한 각 stage에서 보면 모든 processor가 정보 전달에 참여하고 있어서 정보 전달에 필요한 overhead를 최소화할 수 있음을 알 수 있다.

3.4 MPI(Message Passing Interface)

각 processor에서 필요로 하는 정보를 다른 processor로부터 얻거나 혹은 다른 processor가 필요로 하는 정보를 전해주기 위해서는 processor간의 정보 전달을 처리해 줄 수 있는 특별한 기능이 필요하다. 이러한 기능은 일반적인 프로그래밍 언어(C 혹은 Fortran)에는 포함되어 있지 않으며, 특별한 별도의 library를 통하여 구현되어 있다. 이러한 기능을 담당하는 library에는 개개의 hardware에서 최대의 성능을 발휘할 수 있도록 특별히 개발된 것도 있으나(예를 들면, Cray T3E의SHMEM), 본 연구에서는 성능은 다소 떨어지더라도 범용성이 뛰어나 거의 모든 hardware로 이식이 가능한 MPI(Message Passing Interface) library[3]를 이용하였다. MPI library를 이용하여 정보를 주고받기 위해서는 이를 위한 buffer memory 영역을 따로 설정해 두었으며, 필요한 data를 여기에 담거나 뽑아 내어 사용할 수 있도록 하였다. MPI library에는 100여개가 넘는 library function이 포함되어 있으나, 단순히 message를 주고받는 작업에는 몇 개의 기본적인 기능들만이 필요하다. 표 4에는 본 연구에서 사용된 주요 function들 및 각각의 기능을 간략하게 나타내었다.

4. 계산결과

프로그램의 병렬화는 양해법(explicit scheme)을 사용하는 경우 일반적으로 해의 수렴성 및 정확도에 전혀 영향을 미치지 않는다. 즉, 병렬화를 하지 않은 serial solver로써 1,000번의 반복계산에서 원하는 수렴조건을 만족하였다면 processor의 수에 무관하게 병렬화된 solver로도 1,000번의 반복계산으로 동일한 해를 얻어야 한다. 따라서 단순히 반복계산 회수를 결정해 두고, 이에 소요된 계산 시간을 환산하여 병렬처리 효율성을 계산할 수 있다. 그러나 음해법(implicit scheme)이 사용된 경우 병렬처리 효율성의 측정이 쉽지 않다. 우선 각 processor가 각각의 계산 영역에 대한 해를 독립적으로 구하게 되므로, serial solver와 같은 수의 반복계산을 수행하였을 때 두 solver의 해가 같아진다는 보장이 없다. 물론 복잡한 정보 전달 체계를 통하여 같아지도록 만들 수도 있겠지만, 이 경우 프로그램의 대폭적인 수정이 요구될 뿐 아니라, 오히려 계산효율도 떨어지게 될 가능성이 많다. 따라서 이 경우에는 일정한 반복계산 회수를 지정하여 병렬처리 효율을 계산하는 것이 아니라, 해의 수렴조건을 결정하고 그에 소요된 CPU 시간을 이용하여 효율을 산출하는 것이 타당하다. 물론 이 때는 processor 개수에 따라 수렴에 필요한 반복계산 회수가 달라질 것이며, 일반적으로 processor수가 증가함에 따라 수렴된 해를 얻기 위한 반복계산 회수가 증가한다. 이러한 방법은 해의 수렴여부 판단의 일반적인 기준이 되는 방정식 잔류량

Table 4 병렬화에 사용된 주요 MPI function 목록

Function Name	주요 기능
MPI_Init / MPI_Finalize	프로그램의 시작과 끝을 알림
MPI_Send / MPI_Recv	Data를 보내고 받음
MPI_Comm_Size / MPI_Comm_Rank	Processor의 수 및 ID 결정
MPI_Barrier	모든 Processor 작업을 동시화 시킴
MPI_Allreduce	각 Processor의 계산결과를 합산함

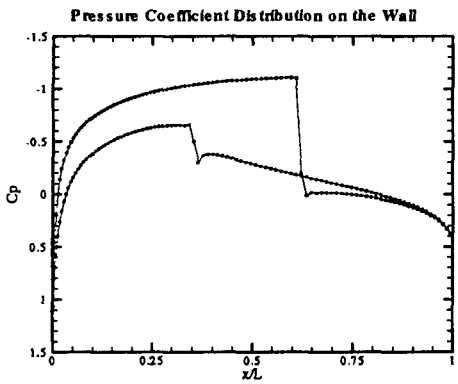

Fig. 3 NACA0012 익형 벽면 압력계수 분포 (M=0.8, α=1.25°)

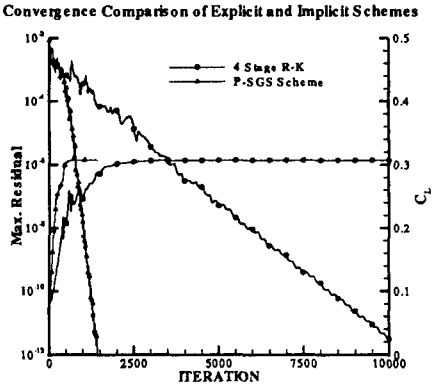

Fig. 4 NACA0012 익형 주위 유동장 계산 수렴 특성 비교 (M=0.8, α=1.25°)

(residual)이 상당히 작은 값까지 줄어드는 경우 가능하나, 그렇지 않은 경우 이 방법을 사용하기가 쉽지 않다. 보통 유동 박리나 복잡한 간섭 현상 등이 있는 경우 유동 방정식의 잔류량은 어느 정도 수렴하다 멈추며, 유동장의 변화(일반적으로 공력 계수의 변화량으로 추정함)는 이보다 더 많은 반복 계산을 수행한 이후에야 멈추게 된다. 공력 계수의 변화로서 수렴 여부를 판단할 수도 있겠으나 방정식의 잔류량이 machine zero 까지 떨어지지 않는 경우에는 해가 반드시 같아진다는 보장도 없으므로, 이와 같은 경우 serial solver와 parallel solver에 같은 수렴 기준을 설정하기가 쉽지 않다. 따라서 본 연구에서는 해의 수렴성이 뛰어나 병렬처리 효율을 결정하기가 용이한 NACA0012 익형 주위의 천음속 비점성 및 난류 유동장 계산에 대하여 그 결과를 살펴보았으며, 해의 수렴이 다소 어려운 다른 문제에 있어서도 비슷한 병렬처리 효율을 보일 수 있을 것으로 생각할 수 있을 것이다.

4.1 천음속 비점성 유동

먼저 NACA0012 익형 주위의 천음속 비점성 유동장 계산에 대한 병렬처리 효율을 살펴보았다. 이 경우는 machine zero 까지 해의 잔류량이 내려가는 완전 수렴된 해를 얻을 수 있으며, 따라서 병렬처리 효율의 산출이 쉬운 경우이다. 계산 조건은 유동 Mach수 0.8, 받음각 1.25° 이며, 그림 3에는 벽면에서의 압력분포를 표시하였다. 이 경우는 격자 형성이 아주 간단하기 때문에 249×50 C-grid를 비정렬격자로 변환하여 사용하였으며, 벽면에서 첫 번째 격자의 거리는 10^{-3} chord로 하였다. Venkatakrishnan의 flux limiter[6]를 사용하여 위아래 벽면 충격파 주위에서의 해의 진동을 없애면서도 수렴된 해를 얻을 수 있었으며, 이에 대한 수렴 곡선이 그림 4에 나타나있다. 이 그림에서 4 stage Runge-Kutta scheme 및 point-SGS scheme의 수렴성을 비교하였으며, point-SGS scheme으로 약 10배정도 빨리 수렴된 해를 얻을 수 있음을 알 수 있다. 수렴된 해는 두 가지 방법에 의한 결과가 완전히 같으므로 그림 3에는 이중 한가지 방법에 의한 결과만을 나타내었다.

다음으로 두 가지 시간적분법에 대한 병렬처리 효율성을 살펴보았다. 계산은 병렬 전용 super-computer인 Cray T3E에서 수행하였으며, 일정한 수렴 조건을 만족할 때까지의 계산 시간으로 병렬처리 효율성을 산출하였다. 먼저 4 stage Runge-Kutta scheme의 병렬처리 결과를 그림 5에 나타내었다. Processor의 개수가 늘어남에 따라 병렬 효율성이 약간 감소하기는 하나, 16개의 processor에서 약 16.9배 정도의 계산시간 단축을 얻을 수 있었다. 여기서 재미있는 현상은 processor의 수에 비하여 병렬처리 효율성 (speed-up factor)이 작은 값을 보이는 것이 일반

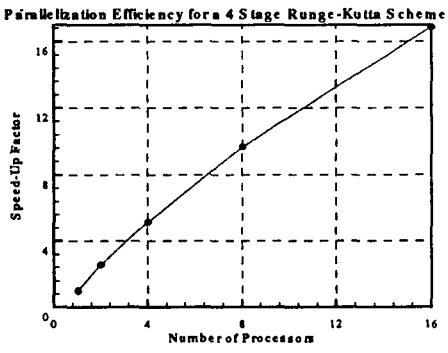

Fig. 5 Explicit scheme에 의한 비점성 천음속 유동장 계산 병렬처리 효율

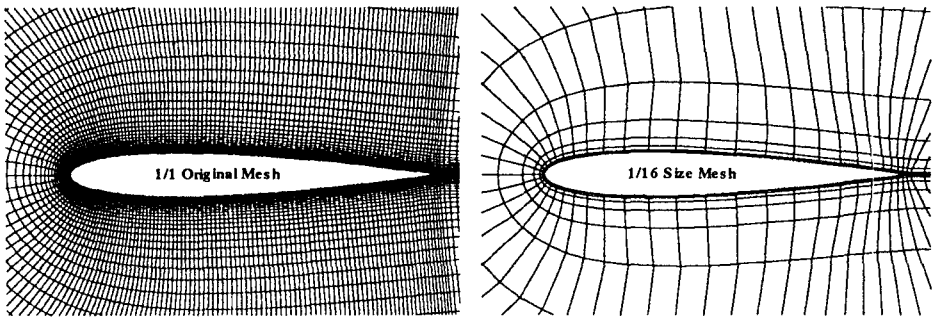

Fig. 6 최초 격자와 1/16 크기 격자 비교

적인데 비하여, 여기서는 오히려 더 큰 값을 보인다는 것이다. 즉, 16개의 processor를 사용하여 병렬계산을 하는 경우 1개의 processor를 사용할 때에 비하여 병렬처리 및 정보전달을 위한 여러 가지 overhead에 의하여 계산시간의 단축이 16배 이하로 이루어지는 것이 대부분이다.

프로그램의 병렬화에 수반되는 overhead를 고려하면 이러한 경우는 거의 예상하기 어려운 것으로, 이는 Cray T3E에 사용된 processor의 특성에 기인한 것으로 보여 진다. 계산에 사용된 Cray T3E의 processor는 DEC Alpha 450MHz RISC chip으로 이론상 최고 성능이 900MFlops에 달한다고 되어 있다. 그러나 이러한 processor의 계산 성능은 프로그램이 얼마나 잘 최적화 되어 있는가에 크게 의존하며, 특히 second level cache memory의 사용도를 높임에 따라 성능이 크게 향상되어 질 수 있다. 물론 이러한 작업은 대부분 compiler에 의해 자동적으로 수행되는 것이 보통이나, 프로그램 크기나 특성에 따른 한계는 있을 것이다. 따라서 병렬처리 효율성(speed-up factor)이 processor의 개수를 증가하는 경우 일단 각 processor에서 처리하는 문제의 크기가 작아짐에 따라 processor의 계산 성능이 증가하였음이 확실하다. 즉 single processor performance가 문제의 크기에 반비례하는 것으로 보여지며, 이를 확인하기 위하여 문제의 크기를 바꾸어 가며 계산을 수행하여 보았다. 그림 6에는 영역 분할 이전의 249×50 C-grid와 16개의 영역으로 분할하였을 때와 거의 같은 크기의 격자를 나타내고 있다. 표 5에는 여러 개의 processor를 사용하기 위하여 영역을 분할하였을 때와 유사한 크기의 격자를 나타내었으며, 이러한 격자에 대한 동일한 천음속 유동장 계산을 수행하였다.

그림 7에는 4 stage Runge-Kutta scheme을 사용하여, 비점성 천음속, 층류 및 난류 유동장을 계산하였을 때 격자수에 따른 processor의 성능을 비교하였다. 종축에 나타낸 speed-up factor는 1/1 mesh의 계산시간을 기준으로 격자 수의 증감량을 감안한 CPU 시간의 비를 나타낸 것이다. 예를 들어, 1000번의 반복 계산에 1/1 mesh로써 1,600sec가 소요되고 동일 회수의 계산에 1/16 mesh로 100sec가 걸렸다면 speed-up factor는 다음과 같이 계산된다.

$$\text{Speed}-\text{Up Factor} = \frac{2,000\ \text{sec}}{100\ \text{sec}} \times \frac{744\ \text{cells}}{12,152\ \text{cells}} = 1.22$$

만약 격자의 크기 감소에 따른 processor의 계산효율 증가가 없다면 speed-up factor의 값은 1이 되어야 하나, 그림에서 보듯이 격자의 크기가 작아질수록 이 값이 급격히 커짐을 알 수 있다. 그리고 이 값은 프로그램의 compile시에 선언해둔 프로그램의 크기에는 상관없이 실제로 계산에 사용된 격자 크기에

Table 5 영역 분할된 격자와 유사한 크기의 격자 성질

	1/1 Mesh	1/2 Mesh	1/4 Mesh	1/8 Mesh	1/16 Mesh
N of Cells	12,152	6,076	2,976	1,488	744
N of Faces	24,577	12,313	6,088	3,056	1,556
N of Nodes	12,425	6,237	3,112	1,568	744

만 의존하는 것으로 나타났다. 이와 같은 현상은 Cray C90과 같은 vector super-computer에서는 나타나지 않는 현상으로 (오히려 vector register 이용 효율이 높아져 문제의 크기가 커질수록 계산 효율이 약간 향상되는 경향을 보임) Cray T3E와 같은 RISC chip을 사용하는 processor의 특징으로 여겨진다. 그러므로 앞에서도 언급한 바와 같이 그림 5에서 보여진 이상적인 경우 보다 큰 병렬처리 효율성(speed-up factor)을 얻을 수 있었던 것은 영역이 분할됨에 따라 각 processor가 처리해야 하는 문제의 크기가 작아진 것에 전적으로 기인한 것임을 알 수 있다. 그림 7을 보면 격자를 16개의 영역으로 나누었을 때 Cray T3E single processor의 계산 효율은 전체를 계산할 때에 비하여 약 2.4배정도 증가됨을 알 수 있으며, 만약 병렬처리에 의한 overhead가 없었다면 그림 5에서 16개 processor에 의한 효율도 16 ×2.4 정도의 값을 가져야 할 것이다. 따라서 Cray T3E에서의 병렬처리 효율을 엄밀히 평가하기 위해서는 전체격자 사용시의 계산시간에 대한 영역분할에 의해 단축된 계산시간만을 기준으로 할 것이 아니라 크기가 줄어든 격자에 대한 single processor performance를 함께 고려해 주어야 할 것이다. 그러나 이러한 방법은 작아진 문제 크기에 해당하는 별도의 격자를 일일이 만들어 계산을 수행해야하는 번거러움이 따르고, 또한 문제의 크기를 작게 함으로써 processor의 효율을 높이는 것 역시 병렬처리의 또 다른 이점으로 볼 수 있으므로 그림 5와 같은 방식의 단순한 병렬처리 효율성 산출 기법을 그대로 사용하였다. 다만 이러한 결과를 해석함에 있어서, 병렬처리 효율의 상당부분이 processor 자체의 계산효율 증가에 기인한다는 사실을 염두에 두어야 할 것이다.

이상으로 explicit scheme을 사용하였을 때의 병렬처리 효율성을 살펴보았으며, 문제의 크기에 따른 Cray T3E processor의 계산성능 향상이 결과에 미치는 영향을 살펴보았다. 다음으로는 음해법인 point-SGS scheme에 대한 병렬처리 효율을 산출하였으며, 그림 8에 그 결과를 나타내었다. point-SGS scheme으로도 앞에서와 같이 이상적인 경우를 뛰어 넘는 병렬처리 효율성을 얻을 수 있었으며, 이는 이미 살펴본 바와 같이 영역 분할로 인한 문제 크기의 축소에 따른 Cray T3E의 single processor performance 향상에 기인한 것임이 확인되었다. 이 그림에 한가지 주목할 점은 processor의 수가 4개인 경우 다른 경우에 비하여 효율성이 비정상적으로 줄어들었다는 점이다. 앞의 explicit Runge-Kutta scheme에서는 동일한 격자로도 이러한 현상을 볼 수 없었기 때문에 이는 point-SGS scheme의 특성에 따른 것으로 추측할 수 있다. point-SGS scheme은 음해법으므로 영역분할에 의해 implicitness가 크게 훼손되는 경우 정해진 수렴조건을 만족하기 위한 반복계산 회수가 증가될 수 있으며, 이 경우 병렬처리 효율성이 현저히 떨어지거나 심한 경우 수렴된 해를 얻기 어려울 수도 있다. 이를 확인하기 위하여 processor의 수가 4인 경우의 영역 분할된 격자를 그림 9에 나타내었으며, 하나의 영역이 익형의 상하에 서로 분리된 격자를 가지고 있음을 알 수 있다.

따라서 이 영역에서 독립적으로 point-SGS scheme에 의한 시간적분을 수행할 때 정보의 전파가 전영역으로 신속히 이루어지지 못함으로써 수렴된 해를 얻기가 어려워질 것임을 알 수 있으며, 이는 그림 9에 나타낸 정해진 수렴조건을 만족하기 위한 반복계산 회수의 비교에서도 잘 나타난다. 이 그림에서 보면, 분할된 영역의 수가 많아질수록 해의 수렴을 위해 많은 수의 반복계산을 필요로 함을 알 수 있으며, 이는 음적시간적분법 사용시의 병렬처리 효율성을 떨어뜨리는 주요한 요인으로 작용한다. 앞에서도 이

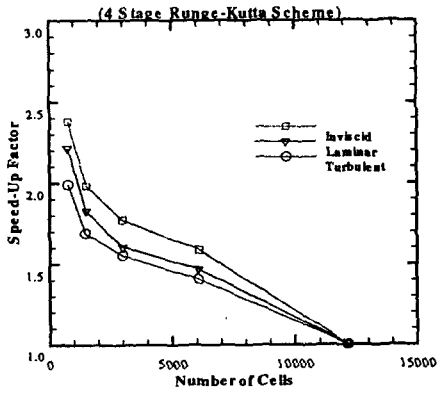

Fig. 7 격자크기에 따른 Cray T3E의 계산효율 (explicit scheme)

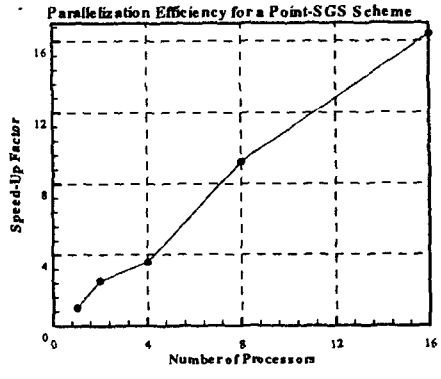

Fig. 8 Implicit scheme에 의한 비점성 천음속 유동장 계산 병렬처리 효율

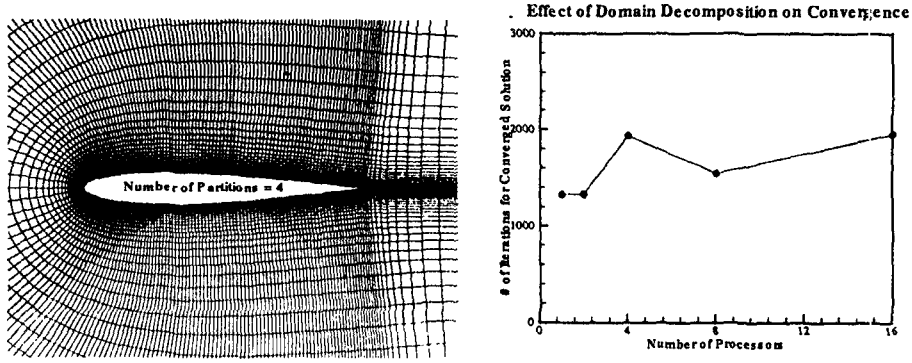

Fig. 9 4개 영역으로 분할된 격자 및 영역 수에 따른 수렴 반복 계산 회수

미 언급한 바와 같이 영역수가 4개 일 때 반복계산 회수가 크게 증가되었음을 알 수 있으며 이로 인해 이 경우의 병렬처리 효율성이 감소되었다. 이러한 결과는 병렬처리를 위한 영역분할에 있어서 중요한 점을 시사한다. 즉, 음해법의 사용시 분할된 개개의 영역 속에 물리적으로 차단된 영역이 존재하거나 그렇지 않다 하더라도 정보의 전파가 어려운 영역이 존재하는 경우 수렴성이 현저히 저하될 수 있다는 점이다. 대부분의 영역분할 기법은 부하의 균등분배(load balancing)에 주안점을 두고 분할된 격자의 수를 같게 하거나, 정보전달의 양을 줄이기 위하여 내부경계면의 크기를 줄이는 두 가지 점을 영역분할 최적화의 목표점으로 삼는다. 따라서 형상이 복잡한 물체의 경우 그림 9에서와 같이 단절된 영역이 만들어질 가능성이 대단히 높다. 본 연구에서도 영역분할을 위해 상용 code인 Fluent 5.0을 사용하였으며 이것은 이러한 점에 대해서는 주의를 기울이지 않는다. 따라서 사용자가 분할된 영역을 확인하여 다른 특성이 다소 나빠지더라도 이러한 현상이 생기지 않도록 주의를 기울여야 할 것이다. 궁극적으로는 시간적분법의 특성에 맞는 영역분할법을 개발하는 것이 적절할 것이며, 이는 다음 연구의 목표로 설정되어 있다.

위의 결과들은 역설적으로 영역분할에 세심한 주의를 기울임으로써 병렬처리 효율성을 상당히 높일 수 있음을 말해주고 있으며, 그림 10은 영역내의 정보 전달이 쉽도록 만들어준 격자와 이에 따른 수렴에 필요한 반복계산 회수를 나타내고 있다. 이 그림을 그림 9와 비교해 보면, 모든 분할된 영역들이 영역 내부에서의 정보 전달이 훨씬 용이하도록 나누어져 있음을 알 수 있으며, 이에 따라 수렴에 필요한 반복

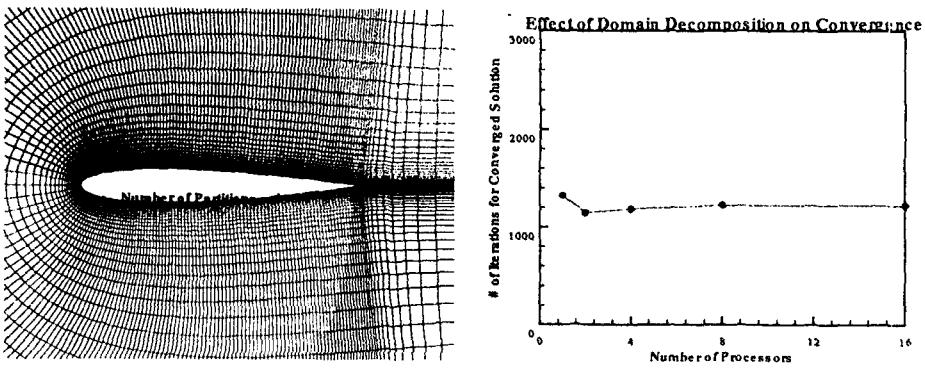

Fig. 10 향상된 영역분할 결과 및 이에 대한 수렴 반복 계산 회수

계산의 회수도 앞에서와는 달리 거의 증가되지 않음을 알 수 있다. 오히려 전체 영역을 계산하는 것에 비하여 영역을 익형의 대칭선을 기준으로 2등분하였을 때 수렴성이 증가되는 것을 알 수 있으며, 이는 이미 언급한 바와 같이 전체 영역에서는 정보의 전달이 익형의 존재로 인하여 상당부분 제약을 받으나 오히려 분할에 의해 각각의 영역은 더 이상적인 형태를 가지기 때문인 것으로 보인다. 그림 11에는 새로운 영역분할에 의한 point-SGS scheme의 병렬처리 효율성을 나타내고 있으며, 그림 8에 비하여 엄청난 효율의 증가를 얻을 수 있음을 알 수 있다. 그림 11에서 보면 16개의 processor를 사용함으로써 약 24배정도 계산속도가 빨라짐을 알 수 있다.(문제의 크기 감소에 의한 processor의 계산 효율을 감안한 이상적 효율은 약 16×2.2 정도임) 따라서 4 stage Runge-Kutta scheme에 비하여 병렬처리에 따른 overhead가 훨씬 줄어든 것으로 나타났으며, 이는 한 번의 반복계산에 소요되는 정보전달의 양이 point-SGS scheme의 경우가 4 stage로 나누어진 Runge-Kutta scheme에 비해 훨씬 작기 때문이다.

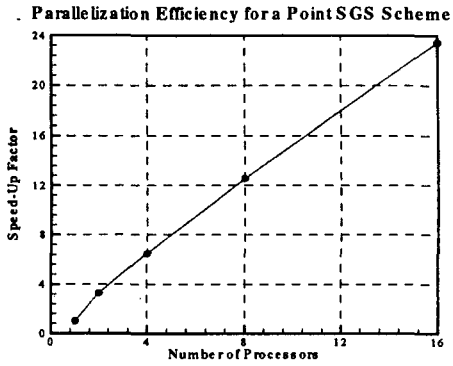

Fig. 11 향상된 영역 분할에 의한 point-SGS scheme의 병렬처리 효율성

3.2 천음속 난류 유동

다음으로 NACA0012 익형 주위의 난류 유동에 대한 병렬 효율을 계산하여 보았다. 유동 조건은 Mach수 0.503, Reynolds수 2.85×10^6, 받음각 6.95° 이며, 계산 격자는 경계층 영역은 사각형으로, 외부는 삼각형으로 채운 혼합격자를 사용하였다. 그림 12에는 8개 영역으로 분할된 격자를 보여주고 있으며, 그림 13에는 벽면에서의 압력분포를 참고문헌 [11]의 실험 결과와 비교하였다. 분할된 각각의 영역이 대부분 약 3개 정도의 다른 영역과 이웃해 있으며, 계산 결과도 실험치와 잘 일치되고 있음을 알 수 있다. 이 경우는 복잡한 유동에 비하여 해의 수렴성이 좋은 경우로 point-SGS scheme의 경우 해의 잔류량이 10^{-8}까지 감소하는 것을 수렴 기준으로 삼았으며, 그림 14에는 1개 영역 및 32개 영역으로 분할된 경우의 수렴특성을 비교하여 나타내었다. 1 영역일 때가 32 영역 보다 수렴성이 약간 좋기는 하나 큰 차이는 없으며, 수렴된 공력계수의 값도 거의 같음을 알 수 있다. 그림 15에는 일정한 반복계산 회수를 기준으로 한 Runge-Kutta scheme의 병렬효율성과 앞의 기준을 적용한 point-SGS scheme의 병렬효율성을 비교하였다. 앞의 비점성 유동 계산과는 달리 processor수가 늘어나도 병렬처리 효율성이 이상적인 경우에서의 값을 크게 넘어서지는 않으며, 이는 그림 7에서 나타낸 바와 같이 각 processor가 담당할 문제 크기의 감소에 따른 계산효율성의 증대가 비점성 문제만큼 크지 않고 또한 전달해야할 정보의 양이 더 늘어났기 때문인 것으로 보인다. 그러나 앞의 경우와 같이 point-SGS scheme이 양해법인 Runge-Kutta

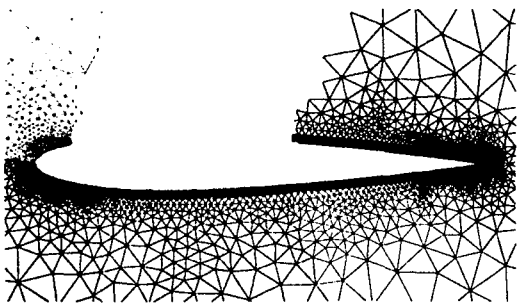

Fig. 12 8개의 영역으로 분할된 NACA0012 익형 주위 난류 유동장 해석을 위한 혼합격자

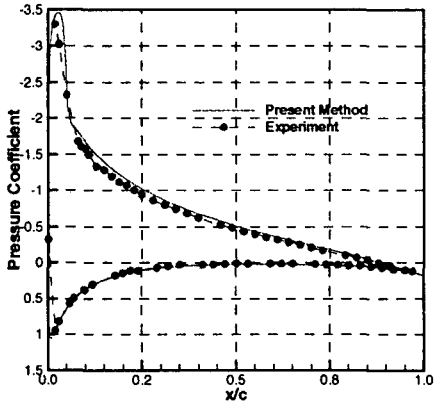

Fig. 13 NACA0012 익형주위 천음속 유동장 해석 결과 비교

scheme 보다 뛰어난 병렬처리 효율성을 보임을 알 수 있다.

4. 결론

본 연구에서는 개발된 혼합격자를 이용한 2차원 난류 유동장 해석 프로그램의 병렬화를 수행하고 이에 수반된 여러 문제점들을 해결하고자 하였다. 먼저 병렬화에 따른 코드 수정요구를 최소화하기 위하여 분할된 각각의 영역에 fringe cell을 추가하는 기법을 채택하였으며, 이를 통하여 serial code에 사용된 subroutine들을 거의 대부분 수정 없이 사용할 수 있게 되었다. 다음으로 각 영역간의 정보전달의 효율성을 기하고 bottleneck이나 deadlock 같은 현상을 방지하기 위하여 최대한의 processor가 동시에 정보전달에 참여 할 수 있는 scheduling 기법을 도입하였다.

병렬화된 code의 병렬처리 효율성을 수렴 조건의 설정이 용이한 NACA0012 익형 주위의 천음속 비점성 및 난류 유동장 계산 문제를 통하여 검증하였으며, 특히 point-SGS scheme의 병렬 특성을 자세히 살펴보았다. 병렬처리 효율성의 산출을 위해 Cray T3E super-computer에서 계산을 수행하였으며, 그 값이 이상적 병렬처리 효율성을 능가할 수 있음을 확인하였다. 그 원인의 분석을 위하여 영역 분할된 격자와 같은 크기의 계산 격자를 만들어 Cray T3E의 single processor performance를 산출하였으며, 격자 크기의 감소에 따라 이 값이 현저히 증가됨을 알 수 있었다. 따라서 이러한 single processor performance의 향상이 병렬처리 overhead에 의한 효율 저하를 훨씬 상회함으로써 이상적인 경우 보다 더 나은 병렬처리 효율성을 얻을 수 있음을 확인하였다.

Point-SGS scheme의 병렬처리에 있어서는, 분할된 격자 내부에서 유동 정도의 흐름이 원활한 격자와 그렇지 못한 격자 사이에 현저한 수렴성의 차이가 있으며, 이에 의해 병렬처리 효율성이 크게 변할 수 있음을 알았다. 따라서 영역분할 기법의 선택이나 개발시 이러한 점이 반드시 고려되어야 하며, 이러한 영역분할 기법의 개발을 다음 연구의 목표중 하나로 설정하였다. 또한 Point-SGS scheme의 병렬처리 효율성이 scheme의 특성을 고려하여 분할된 격자를 사용하였을 때 4 stage Runge-Kutta scheme에 비해 훨씬 크다는 점을 알 수 있었으며, 대부분의 implicit scheme들의 특징인 영역분할에 따른 수렴성의 저하도 크지 않음을 알 수 있었다. 즉, point-SGS scheme을 그 특성에 맞는 영역분할법과 결합하여 사용됨으로써 영역분할에 의한 수렴성의 저하를 최소화하고 뛰어난 병렬처리 효율성을 얻을 수 있었다.

후기

본 연구는 과학기술부 "초고속 컴퓨터 기반 소프트웨어 및 응용기술 개발" 사업의 세부과제인 "병렬 3차원 CFD 소프트웨어 개발" 과제의 협동연구과제로 수행되었으며, 병렬 계산에는 연구개발정보센터(KORDIC) 슈퍼컴퓨팅사업단의 Cray T3E super-computer를 사용하였습니다.

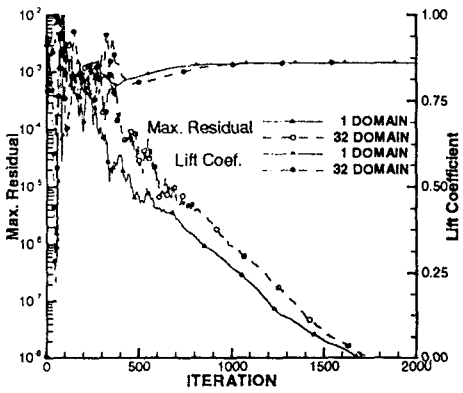

Fig. 14 Implicit scheme에 의한 비점성 천음속 유동장 계산 병렬처리 효율

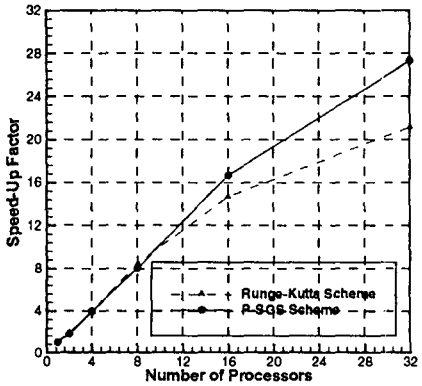

Fig. 15 Implicit scheme에 의한 비점성 천음속 유동장 계산 병렬처리 효율

참고문헌

[1] 박승오 외, "병렬 3차원 Computational Fluid Dynamics 소프트웨어 개발," 과학기술부 97-NF-03-01-A-06, 1999. 9.

[2] 옥호남, 박승오, "혼합격자를 이용한 2차원 난류 유동장 해석 프로그램 개발," 한국항공우주학회 1999년도 추계학술발표회 논문집, 1999.

[3] MPI: A Message-Passing Interface Standard, Message Passing Interface Forum, June 1995.

[4] Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," Journal of Computational Physics, Vol. 43, No. 7, pp. 357-372, 1981.

[5] Barth, T. J. and Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, 1989.

[6] Venkatakrishnan, V., "On the Convergence of Limiters and Convergence to Steady State Solutions," AIAA Paper 93-0880, 1993.

[7] Ok, H. and Eberhardt, D. S., "Development of an Unsteady Incompressible Navier-Stokes Solver and Its Application to the Computations of Separated Flows," AIAA Paper 91-3266, 1991.

[8] Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92-0439, 1992.

[9] Shostko, A. and Löhner, R., "A Parallel Adaptive Finite Element Flow solver For Transient Problems," AIAA Paper 95-1662-CP, 1995.

[10] Fluent 5 User's Guide, Fluent Inc., July 1998.

[11] Thibert, J. J., Grandjacques, M., "Experimental Data Base for Computer Program Assessment - Report of the Fluid Dynamics Panel Working Group 04," AGARD-AR-138, p.p. A1-1 - A1-36, May 1979.