병렬처리를 이용한 대규모 동적 시스템의 최적제어

Optimal Control of Large-Scale Dynamic Systems using Parallel Processing

박 기 홍
(Kihong Park)

Abstract : In this study, a parallel algorithm has been developed that can quickly solve the optimal control problem of large-scale dynamic systems. The algorithm adopts the sequential quadratic programming methods and achieves domain decomposition-type parallelism in computing sensitivities for search direction computation. A silicon wafer thermal process problem has been solved using the algorithm, and a parallel efficiency of 45% has been achieved with 16 processors. Practical methods have also been investigated in this study as a way to further speed up the computation time.

Keywords : optimal control, large-scale dynamic systems, parallel processing, sequential quadratic programming

Ⅰ. 서론

시스템의 최적제어란 시스템의 구속조건을 만족시키면서 주어진 비용함수를 최소화하는 제어변수의 값을 찾는 문제로 공학이나 과학의 여러분야에서 찾을 수 있다. 동적 시스템의 경우 그 특성은 상미분방정식이나 편미분방정식의 구속조건으로 나타나는데 특히 편미분방정식의 경우 공간좌표계에 대한 이산화를 거치면 많은 상미분방정식으로 변환된다. 본 연구에서는 비선형 프로그래밍 기법을 사용하여 이와 같은 대규모 동적시스템의 최적제어를 위한 병렬 알고리즘을 개발하고자 한다.

병렬처리를 이용한 최적제어 알고리즘에 관한 연구는 70년대부터 싹트기 시작하다가[1][2], 병렬처리 수퍼컴퓨터가 본격적으로 상용화되기 시작한 90년대에 들어 매우 활발해졌다. Betts와 Huffman은 궤도최적화 알고리즘에서 편미분값들을 유한편차에 의해 구했으며[3], 이 과정을 병렬처리한 프로그램을 8개의 프로세서를 지닌 BBN GP1000 컴퓨터에서 수행하여 2배에서 5배에 이르는 연산속도의 증가를 얻었다. Wright은 quadratic programming 문제의 크기를 연쇄적으로 줄이고 마지막으로 남는 문제를 다이나믹 프로그래밍 알고리즘을 써서 푸는 병렬 알고리즘을 개발하였다[4]. 그는 8개의 프로세서를 가진 Alliant FX/8 컴퓨터로 약 3배에서 5배에 이르는 연산속도의 증가를 보였다.

본 연구의 목적은 대규모 동적 시스템의 최적제어 문제를 빠르게 풀기 위한 병렬 알고리즘을 개발하는 것이다. 이를 위해 최적화 알고리즘이 필요로 하는 민감도 계산을 영역분해 방식으로 병렬처리하였으며 연산시간을 줄이기 위한 여러 실용적인 기법에 대해 연구하였다.

Ⅱ. 연구 방법 및 이론

1. 최적제어 문제

동적 시스템의 최적제어 문제는 다음과 같이 수학적으로 정의될 수 있다.

$$\text{find}: \ u(t) \text{ and } v(t) \text{ for } t_0 \le t \le t_f \tag{1}$$

$$\text{to minimize}: \ \Phi = \int_{t_0}^{t_f} L(t, v(t), u(t))dt + V(v(t_f)) \tag{2}$$

$$\text{subject to}: \ v'(t) = f(t, v(t), u(t)) \tag{3}$$

$$h(t, v(t), u(t)) \ge 0 \tag{4}$$

윗 식에서 $v(t)$는 상태변수 벡터, $u(t)$는 제어변수 벡터, t_0는 초기시간, t_f는 말기시간을 나타내며, ( )'은 시간에 대한 미분을 나타낸다. (2)는 비용함수이다. (3), (4)는 구속조건들로서 (3)은 시스템의 동특성을 나타내는 미분방정식이고 (4)는 그 밖의 부등식형 구속조건이다. 등식의 구속조건은 2개의 부등식 구속조건으로 나타내어질 수 있다.

이산화. 비선형 프로그래밍은 유한한 개수의 파라미터를 가진 최적화문제를 푸는 기법으로 이를 최적제어에 사용하기 위해서는 먼저 연속시간 상에서 정의된 (1)-(4)의 문제를 이산화해야 한다. 이를 위해 전체 시간영역 $[t_0, t_f]$를 N_t개의 시구간으로 나눌 수 있다. k번째 시구간의 시작점 t_k는 다음과 같이 정의된다.

$$t_k = t_0 + k\Delta t, \quad k = 0, \ldots, N_t - 1 \tag{5}$$

$$\Delta t = \frac{t_f - t_0}{N_t} \tag{6}$$

최적제어 문제의 이산화란 변수들과 함수들의 이산화를 의미한다. 먼저 제어변수의 경우 각각의 시구간에서 $u(t)$를 일정한 차수의 다항식 함수로 만들므로써 이산화시킬 수 있다.

$$\hat{u}_k(t) = u_{k,0} + u_{k,1}(t - t_k) + u_{k,2}(t - t_k)^2,$$

$$\text{for } t \in [t_k, t_{k+1}] \tag{7}$$

접수일자 : 1998. 11. 11., 수정완료 : 1999. 3. 31.
박기홍 : 국민대학교 기계자동차공학부
※ 이 연구는 1996년도 한국과학재단 연구비지원에 의한 결과입니다(과제번호: 961-1001-004-1).

(7)은 t의 2차 다항식인데 이보다 고차나 저차의 다항식으로도 모델링할 수 있다. (7)과 같은 모델링으로 k번째 시구간에서의 제어변수의 변화는 계수 $u_{k,0}$, $u_{k,1}$, $u_{k,2}$에 의해 결정되어 제어변수는 유한한 개수의 파라미터로 나타낼 수 있게 된다. 이에 따라 다음과 같은 이산화된 제어변수를 정의할 수 있다.

$$u_k = [u_{k,0}^T, u_{k,1}^T, u_{k,2}^T]^T, \quad k = 0, \ldots, N_t - 1 \tag{8}$$

제어변수가 연속적으로 변해야 하는 경우에는 인접한 시구간 사이에 (9)의 구속조건을 사용할 수 있다.

$$u_{k+1,0} = u_{k,0} + u_{k,1}\Delta t + u_{k,2}(\Delta t)^2 \tag{9}$$

상태변수의 **이산화**를 위해 연속시간 영역에서의 $v(t)$를 다음의 **물리량** v_0, v_1,..., v_{N_t}로 나타낼 수 있다.

$$v_k = v(t_k), \quad k = 0, \ldots, N_t \tag{10}$$

제어변수와 상태변수의 이산화에 따라 (1)-(4)의 연속시간 최적제어 문제는 다음과 같은 이산시간 영역에서의 최적제어 문제로 변환된다.

$$\text{find}: \ v_0, u_0, v_1, \ldots, u_{N_t-1}, v_{N_t} \tag{11}$$

$$\text{to minimize}: \ \Phi = \sum_{k=0}^{N_t-1} L_k(k, v_k, u_k) + V(v_{N_t}) \tag{12}$$

$$\text{subject to}: \ v_{k+1} = f_k(k, v_k, u_k), \quad k = 0, \ldots, N_t - 1 \tag{13}$$

$$h_k(k, v_k, u_k) \geq 0, \quad k = 0, \ldots, N_t \tag{14}$$

다음은 윗 식에 있는 함수들의 이산화 과정을 보여준다. 먼저 함수 f_k는 다음과 같다.

$$f_k(k, v_k, u_k) \equiv \hat{v}_k(t_{k+1}) \text{ where } \hat{v}_k(t) \text{ satisfies}:$$

$$\hat{v}_k'(t) = f(t, \hat{v}_k(t), \hat{u}_k(t)), \quad \hat{v}_k(t_k) = v_k \tag{15}$$

변수 $\hat{v}_k(t)$는 (15)의 초기조건 문제의 해이다.

(10)의 상태변수 이산화를 위해 시스템 운동방정식을 초기시간 t_0로부터 t_f까지 적분해야 한다면 이는 순차적인 수치계산이며 따라서 병렬화는 어려워진다. 대규모 동적 시스템의 미분방정식을 적분하는데는 상당한 시간이 걸리기 때문에 전체 계산시간을 줄이기 위해서는 이 부분의 병렬화가 필수적이다. 본 연구에서 설계된 최적제어 알고리즘은 $v_0, v_1, \ldots, v_{N_t}$와 $u_0, u_1, \ldots, u_{N_t-1}$을 모두 독립적인 최적화 변수로 취급하기 때문에 임의로 선택된 $v_1, v_2, \ldots, v_{N_t}$의 값들로부터 최적해를 찾을 수 있다. 이는 각 시구간에서 인접 시구간과 상관없이 (15)를 적분할 수 있게 만들어 병렬 계산을 가능하게 한다. 따라서 알고리즘이 최적해를 찾아가는 과정중에는 $\hat{v}_k(t)$와 $v_k(t)$가 다를 수 있는데 그림 1은 상태변수의 개수가 1인 경우 $\hat{v}_k(t)$의 형태를 보여준다.

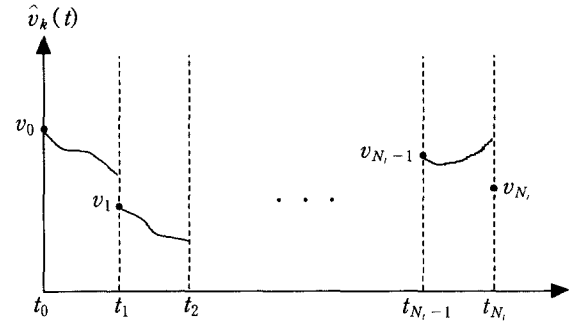


그림 1. 최적화 초기의 이산화된 상태변수 형태.
Fig. 1. Time history of a discretized state variable at the early stage of optimization process.

그림 1은 최적화 초기에 있어서 상태변수의 가능한 모양을 나타내는데 알고리즘이 최적해에 이르면 (15)와 같은 이산화 함수 f_k의 정의로 $v_1, v_2, \ldots, v_{N_t}$는 (3)의 시스템 운동방정식을 만족하게 되고, 따라서 $\hat{v}_k(t)$는 $v_k(t)$와 동일한 연속함수가 된다. 이같은 성질은 비선형성이 심한 시스템을 다루는 경우 특히 도움이 된다.

(12)의 함수 L_k는 다음과 같이 정의하였다.

$$L_k(k, v_k, u_k) \equiv \int_{t_k}^{t_{k+1}} L(t, \hat{v}_k(t), \hat{u}_k(t))\, dt \tag{16}$$

이 값은 다음과 같은 초기조건 문제로 변환하여 구할 수 있다.

$$L_k(k, v_k, u_k) = \hat{L}_k(t_{k+1}) \text{ where } \hat{L}_k(t) \text{ satisfies}:$$

$$\hat{L}_k'(t) = L(t, \hat{v}_k(t), \hat{u}_k(t)), \quad \hat{L}_k(t_k) = 0 \tag{17}$$

최적제어 문제의 성격상 L_k의 값을 이처럼 정확하게 구하지 않아도 되는 경우라면 (16)의 적분을 사다리꼴 법칙과 같은 근사법으로 구하는 것도 고려할 만하다.

(14)의 이산화 함수 h_k는 (4)의 함수 h에 k번째 시구간 초기의 이산화 변수를 대입한 값을 갖도록 정의하였다.

$$h_k(k, v_k, u_k) \equiv h(t_k, v_k, u_{k,0}) \tag{18}$$

(18)과 같은 이산화 함수 h_k의 정의로 (4)의 구속조건은 시구간의 경계점을 제외한 곳에서 만족되지 않을 수 있다. 이같은 문제에 대해서는 몇 가지 해결책이 제시될 수 있다. 먼저 시구간의 수 N_t를 늘리는 것이다. 그러나 이 방법은 전체 변수의 수를 증가시켜 최적화 문제를 더욱 대형화시키므로 신중히 선택하여야 한다. 둘째는 (4)의 오른쪽 항의 값을 인위적으로 0보다 크게 잡는 것이다. 이렇게 하면 경계점 $t_0, t_1, \ldots, t_{N_t}$에서 부등식은 보다 여유있게 만족되어 시구간 경계의 사이점에서 부등식이 만족되지 않을 가능성은 줄게 된다. 세째는 구속조건이 제어변수만의 함수인 경우에 해당되는 것으로, 각각의 시구간을 더 잘게 나눈 점에서 해당 구속조건을 추가하는 것이다. 이 방법은 전체 변수의 수는 증가시키지 않

고 다만 구속조건의 수를 증가시킨다.

2. 최적화 알고리즘

최적화 문제를 일반적인 형태로 나타내면 다음과 같다.

$$\text{find}: \ z \in R^{n_z} \tag{19}$$

$$\text{to minimize}: \ \Phi(z) \tag{20}$$

$$\text{subject to}: \ c(z) \geq 0 \tag{21}$$

본 연구에서는 최적화 알고리즘으로 Sequential Quadratic Programming(SQP) 알고리즘[5]을 사용하였다. (21)의 구속조건에 대한 Lagrange multiplier를 λ라 하면 SQP 알고리즘의 기본적인 흐름도는 다음과 같다.

[1] 최적해와 그 점에서의 λ값을 예측한다(z_i, λ_i).

[2] 현재의 반복점이 최적조건을 만족하면, 알고리즘을 끝낸다.

[3] 탐색방향 Δz를 계산한다.

[4] 새 반복점으로 옮기고($z_i \leftarrow z_i + \alpha \Delta z$), 새 반복점에서의 λ를 계산한다.

[5] 단계 [2]로 간다.

최적해를 z^*, 최적해에서의 Lagrange multiplier를 λ^*라고 하면, 위 알고리즘은 반복적으로 최적해 (z^*, λ^*)에 수렴해 나간다. (19)-(21)의 문제를 위한 SQP 최적화 알고리즘에서 Lagrangian 함수와 modified Lagrangian 함수는 다음과 같이 정의된다.

$$\Psi(z, \lambda) = \Phi(z) - c(z)^T \lambda \tag{22}$$

$$\Psi_M(z, z_i, \lambda) = \Phi(z) - (c(z) - c_L(z, z_i))^T \lambda \tag{23}$$

윗 식에서 $c_L(z, z_i)$는 $c(z)$의 z_i에서의 선형화된 함수로 (24)와 같이 나타내어진다. (24)의 $J(z)$는 Jacobian으로 $c(z)$의 1차 민감도를 나타낸다.

$$c_L(z, z_i) = c(z_i) + J(z_i)(z - z_i) \tag{24}$$

SQP 알고리즘에 있어 가장 핵심이 되는 부분은 탐색방향 Δz의 계산이다. 탐색방향은 다음과 같은 Quadratic Programming(QP) 최적화 문제의 해이다.

$$\text{find}: \ \Delta z \in R^{n_z}, \ \text{to minimize}: \tag{25}$$

$$\Phi(z_i) + g(z_i)^T \Delta z + \frac{1}{2} \Delta z^T H_L(z_i, \lambda_i) \Delta z \tag{26}$$

$$\text{subject to}: \ c(z_i) + J(z_i) \Delta z \geq 0 \tag{27}$$

위 식에서 함수 g는 gradient 벡터로 비용함수의 1차 민감도 값이다. $H_L(z_i, \lambda_i)$는 반복점 (z_i, λ_i)에서의 Ψ_M의 Hessian 행렬로 다음과 같다.

$$H_L(z_i, \lambda_i) = H_\Phi(z_i) + \sum_j (\lambda_{i_j} H_{c_j}(z_i)) \tag{28}$$

$$H_\Phi(z) = \frac{\partial^2 \Phi(z)}{\partial z^2}, \quad H_{c_j}(z) = \frac{\partial^2 c_j(z)}{\partial z^2} \tag{29}$$

윗 식에서 λ_{i_j}와 $c_j(z)$는 각각 벡터 λ_i와 $c(z)$의 j번째 요소를 가리킨다.

탐색방향은 (25)-(27)과 같은 최적화 문제의 해가 되므로 이 또한 반복적으로 찾아져야 한다. 결국 SQP 알고리즘은 z^*의 값을 찾는 주반복(major iteration) 과정과 주반복의 매 단계에서 Δz의 값을 찾는 부반복(minor iteration)의 두 과정으로 이루어진다. SQP 알고리즘은 현재의 주반복점 (z_i, λ_i)에서 다음과 같은 최적조건이 만족되면 반복과정을 종료한다.

$$g(z_i) = J(z_i)^T \lambda_i, \ \lambda_i \geq 0 \tag{30}$$

$$c(z_i) \geq 0, \ c(z_i)^T \lambda_i = 0 \tag{31}$$

SQP 알고리즘은 (25)-(27)에서 보는 바와 같이 주반복점을 옮길 때마다 탐색방향을 구하기 위해 비용함수 및 구속조건 함수를 계산해야 하고 또한 1차 민감도 값인 $g(z_i)$와 $J(z_i)$, 2차 민감도인 $H_L(z_i, \lambda_i)$를 구해야 한다. 이 중 민감도는 알고리즘 전체 연산량의 반 이상을 차지할 만큼 많은 계산량을 요구한다. 이에 따라 본 연구에서는 계산시간을 줄이고자 민감도 계산을 병렬화하였으며 이에 대한 내용은 다음 절에서 소개된다.

3. 병렬 최적제어 알고리즘

대규모 최적제어 문제의 해를 계산하는데는 많은 시간이 소요된다. 이 중 가장 많은 계산을 필요로 하는 곳은 탐색방향 QP를 위해 민감도를 계산하는 수치적분 부분이다. 본 연구에서는 2.1절에서 소개된 최적제어 문제의 영역분해 및 변수와 함수의 이산화를 바탕으로, 민감도 계산을 병렬화하여 최적제어 알고리즘의 연산시간을 줄이고자 하였다. 이 절에서는 병렬화된 민감도 계산 방법과 병렬화에 따른 프로세서간의 메시지 교환 방법에 대해 설명한다.

민감도 계산의 병렬화. 2절에서 탐색방향 QP는 (z_i, λ_i)에서의 민감도인 벡터 $g(z)$와 행렬 $J(z)$ 및 H_L (z, λ)로 구성됨을 보았다. (11)-(14)와 같은 이산화된 최적제어 문제에서 이들은 다음과 같이 나타난다. 먼저 $g(z)$는 각 시구간에서의 다음의 g_{u_k}와 g_{v_k}로 이루어진다.

$$g_{u_k} = \frac{\partial L_k(k, v_k, u_k)}{\partial u_k} \tag{32}$$

$$g_{v_k} = \begin{cases} \dfrac{\partial L_k(k, v_k, u_k)}{\partial v_k}, & k \neq N_t \\ \dfrac{\partial V(v_{N_t})}{\partial v_{N_t}}, & k = N_t \end{cases} \tag{33}$$

Jacobian 행렬 $J(z)$는 다음의 J_{u_k}와 J_{v_k}로 이루어진다.

$$J_{u_k} = \begin{bmatrix} J_{fu_k} \\ J_{hu_k} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_k(k, v_k, u_k)}{\partial u_k} \\ \dfrac{\partial h_k(k, v_k, u_k)}{\partial u_k} \end{bmatrix} \tag{34}$$

$$J_{v_k} = \begin{bmatrix} J_{fv_k} \\ J_{hv_k} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_k(k, v_k, u_k)}{\partial v_k} \\ \dfrac{\partial h_k(k, v_k, u_k)}{\partial v_k} \end{bmatrix} \tag{35}$$

Hessian 행렬 $H_L(z,\lambda)$는 다음의 H_{uu_k}, H_{uv_k}, H_{vv_k}의 행렬들로 구성된다.

$$H_{uu_k} = \frac{\partial^2 \Psi}{\partial u_k^2},\ H_{uv_k} = \frac{\partial}{\partial v_k}\left(\frac{\partial \Psi}{\partial u_k}\right),\ H_{vv_k} = \frac{\partial^2 \Psi}{\partial v_k^2} \quad (36)$$

다음은 위의 민감도 값들을 수치적으로 구하기 위해 본 연구에서 사용된 방법을 소개한다. 먼저 (34)-(35)의 J_{hu_k}와 J_{hv_k}는 해석적으로 계산될 수 있음을 알 수 있다. (34)의 J_{fu_k}는 다음과 같이 나타낼 수 있다.

$$J_{fu_k} = \frac{\partial \hat{v}_k(t_{k+1})}{\partial u_k} \quad (37)$$

위의 J_{fu_k}는 다음과 같은 등가의 초기조건 문제로 변환하여 구할 수 있다.

$$\hat{J}_{fu_k}(t) = \frac{\partial \hat{v}_k(t)}{\partial u_k},\ \text{where}\ \hat{J}_{fu_k}(t)\ \text{satisfies}:$$

$$\frac{d}{dt}\left[\hat{J}_{fu_k}(t)\right] = \left[\frac{\partial f(t,v,u)}{\partial v}\right]_k \left[\hat{J}_{fu_k}(t)\right] + \left[\frac{\partial f(t,v,u)}{\partial u}\right]_k,\ \left[\hat{J}_{fu_k}(t_k)\right] = 0 \quad (38)$$

(38)에서 $[\,]_k$는 []안의 값을 계산한 후 $(t,v,u)=(t, \hat{v}_k(t), \hat{u}_k(t))$를 대입함을 의미한다. 같은 방법으로 하면 J_{fv_k}도 초기조건 문제의 해로써 구할 수 있다.

(32)의 g_{u_k}는 (16)을 u_k에 대해 편미분하여 얻어지는 적분인데 이 적분은 다음과 같은 등가의 초기조건 문제로 변환할 수 있다.

$$g_{u_k} = \hat{g}_{u_k}(t_{k+1}),\ \text{where}\ \hat{g}_{u_k}(t)\ \text{satisfies}:$$

$$\hat{g}_{u_k}'(t) = \left[\hat{J}_{fu_k}(t)\right]^T \left[\frac{\partial L(t,v,u)}{\partial v}\right]_k^T + \left[\frac{\partial L(t,v,u)}{\partial u}\right]_k^T,\ \hat{g}_{u_k}(t_k) = 0 \quad (39)$$

같은 방법으로 g_{v_k}도 또한 구할 수 있다.

(36)의 행렬 H_{uu_k}, H_{uv_k}, H_{vv_k} 또한 위와 같이 초기조건 문제로 변화하여 수치적으로 구할 수 있지만 이렇게 할 경우 계산량이 과도하게 많아지는 단점이 있다. 따라서 본 연구에서는 (16)의 적분을 사다리꼴 법칙으로 근사화하여 구현하는 방법만을 시도하였다. 이렇게 하면 (22)의 Ψ 함수에는 미분이나 적분이 전혀 포함되지 않아 H_{uu_k}, H_{uv_k}, H_{vv_k}를 해석적으로 구할 수 있다.

이상에서 소개된 민감도 계산방법을 살펴보면 시구간 k에서의 모든 민감도 값은 같은 시구간의 변수값에만 의존하고 다른 시구간의 변수로부터 영향을 받지 않음을 알 수 있다. 따라서 병렬컴퓨터가 M개의 프로세서를 가지는 경우 한 개의 프로세서로 하여금 N_t/M개의 시구간의 민감도 연산을 위한 수치적분을 담당하게 함으로써 민감도 계산을 병렬화할 수 있다.

본 연구에서는 민감도 계산에 필요한 수치적분을 위해 미분대수방정식을 풀기 위해 개발된 DASPKSO를 사용하였다. DASPKSO는 민감도를 Newton method를 사용한 반복적 방법으로 계산하면서 수치 효율을 최대한 얻도록 설계되어 미분방정식을 위한 연산에 약간의 추가적 연산만으로 민감도를 구할 수 있으며[6] 이러한 성능은 본 연구의 최적화 알고리즘 개발에 있어 매우 유용하다.

메시지 교환. 병렬 프로세서들은 수행하는 수치연산의 종류에 따라 주(master)프로세서와 종(slave)프로세서로 구분지어질 수 있다. 병렬 최적제어 알고리즘에 있어서 M개의 종프로세서들은 각각 할당된 시구간의 수치적분을 수행하며 1개의 주프로세서는 수치적분을 제외한 모든 최적화 연산을 수행한다. 컴퓨터 하드웨어에 따라 주프로세서를 독립적으로 둘 수도 있고, 종프로세서 중 하나가 주프로세서의 기능까지 맡아보도록 할 수도 있다.

병렬 최적제어 프로그램 수행에 필요한 메시지 교환을 살펴보면 다음과 같다. 먼저 각각의 시구간에서의 수치적분은 다른 시구간의 변수 값으로부터 전혀 영향을 안 받도록 설계되었기 때문에 종프로세서들 사이에는 데이터 교환이 필요없다. 주프로세서를 살펴보면, 최적화 알고리즘에서 주반복점이 새로 바뀌고 난 후 그 점에서의 탐색방향 계산을 위해 새 반복점을 시구간별로 분해하여 데이터를 해당 종프로세서에 전해준다. 종프로세서들은 이를 받아 자신이 담당하는 시구간의 함수값과 민감도 값을 계산한다. 계산이 끝나면 종프로세서들은 이들 값을 다시 주프로세서에 보낸다. 주프로세서에서는 이들을 모아 탐색방향을 계산하고 이 방향을 따라 새로운 반복점으로 옮겨간다. 그림 2는 4개의 종프로세서를 사용하여 $N_t = 7$로 이산화된 최적제어 문제를 풀 때 프로세서 사이에 일어나는 메시지 교환을 도식적으로 나타낸다.

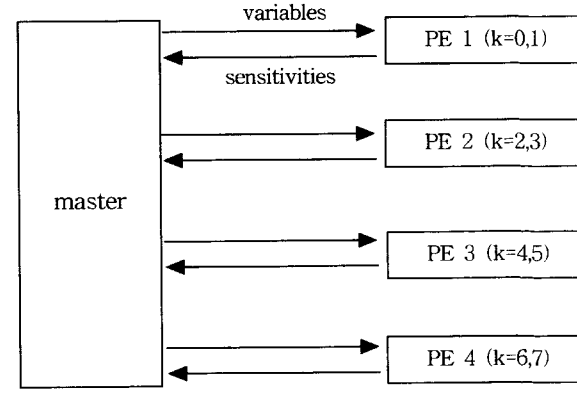


그림 2. 병렬 최적제어 알고리즘의 메시지 교환.
Fig. 2. Message passing pattern in parallel optimal control algorithm.

그림 2와 같은 메시지 교환은 최적해에 이를 때까지 계속해서 반복된다. 또한 위의 상황은 주반복 과정 뿐 아니라 탐색방향의 QP 문제를 푸는 부반복 과정에서도 필요하기 때문에 주반복 1회가 수행되는 동안 그림 2와 같은 데이터 교환이 여러번 이루어져야 한다. 메시지 교

환을 위해서 본 연구에서는 이식성(portability)을 고려하여 PVM(Parallel Virtual Machine)[7]을 사용하였다.

4. 연산시간의 감소를 위한 추가적 연구

본 연구에서는 병렬화와 더불어 최적화 연산시간을 줄이기 위한 실용적인 방법에 대해서도 고찰하였다.

그 첫째가 효율적인 메시지 교환에 관한 연구이다. 병렬 최적제어 알고리즘 상에서 수치적분을 M개의 프로세서에서 분산계산하면 연산시간은 M배 빨라진다. 하지만 병렬화로 인해 메시지 교환이 필요하게 되기 때문에 메시지 교환에 드는 시간만큼 전체 연산시간은 늘어난다. 본 연구에서는 메시지 교환으로 인한 시간지연을 줄이기 위해 효율적인 방법의 메시지 교환을 시도하였다. 종프로세서들이 독립적인 연산을 마치면 데이터를 주프로세서에 전송해야 하는데 이 때 병목현상이 발생한다. 주프로세서가 한 번에 한 개의 프로세서로부터만 데이터를 전송받을 수 있기 때문이다. 하지만 데이터를 종프로세서에서 먼저 집합한 다음 주프로세서에 보내면 위의 병목을 완화시킬 수 있다. 예를 들어 4개의 종프로세서가 있는 경우 프로세서 2가 프로세서 1에게 데이터를 전송하고 이와 동시에 프로세서 4가 프로세서 3에게 데이터를 전송한다. 그 다음 프로세서 3은 자신이 가지고 있는 데이터와 프로세서 4로부터 받은 데이터를 프로세서 1에게 전송한다. 그 다음 프로세서 1은 모든 종프로세서에서 모여진 데이터를 주프로세서로 전송한다. 물론 각 단계마다 데이터의 량이 2배씩 증가하지만 이에 따른 전송시간의 증가는 상대적으로 크지 않다. 이러한 메시지 교환방법의 효율은 프로세서의 수가 많을수록 증가한다는 것을 알 수 있다.

연산시간의 감소를 위한 두번째 노력으로 Jacobian 행렬을 재사용하는 것이 연구되었다. 만약 최적제어 문제의 구속조건이 선형이라면 Jacobian은 항상 일정하다. 만약 구속조건이 비선형이지만 비선형성이 약한 경우라면 주반복점이 바뀌어도 Jacobian은 크게 다르지 않을 것이다. Jacobian은 탐색방향을 계산하는데 쓰이기 때문에 이 값이 틀리면 최적화 알고리즘은 본래 의도한 방향과 다른 방향으로 움직이게 된다. 최적화 문제의 비선형성이 아주 심하지 않은 이상, 약간 틀린 Jacobian에 기초한 탐색방향은 적어도 비용함수 값을 줄일 수 있는 감소방향(descent direction)일 가능성이 크며 뉴톤방향과 크게 다르지 않을 가능성 또한 높다. 이러한 사실에 착안하여 본 연구에서는 알고리즘 첫 부분에서 계산된 Jacobian을 그 이후 반복점에서도 재사용하는 방법을 시도하여 고무적인 결과를 얻었으며 이에 대한 내용은 3장에서 다룰 것이다.

연산시간의 감소를 위한 세번째 노력으로 효율적인 예측해 추정 방법이 연구되었다. 최적화 알고리즘은 예측해로부터 출발하여 최적해를 찾아나가는데 예측해를 잘 선택할 수 있다면 전체 최적화 연산시간의 많은 절감을 얻을 수 있다. 본 연구에서는 좋은 예측해를 찾는 방법으로 다음과 같은 점을 고찰하였다. 최적제어 문제의 이산화를 위해 시간영역을 N_t개로 나누는데 N_t가 큰 경우에는 먼저 $N_t/2$개의 시구간으로 이산화된 문제를 풀 수 있다. $N_t/2$개의 시구간으로 이산화된 문제는 N_t개의 시구간으로 이산화된 문제에 비해 민감도 계산량이 적다. 또한 최적화 알고리즘은 보통 변수의 수가 작은 문제의 해를 보다 빠르게 찾는다. $N_t/2$개의 시구간 최적제어 문제의 해를 찾고 나면 각 시구간을 둘로 나누고 보간에 의해 상태변수와 제어변수를 각 시구간 내의 두 시구간에 대해 새롭게 정의할 수 있다. 이렇게 두 배로 크기가 늘어난 상태변수와 제어변수는 N_t개의 시구간 최적제어 문제의 예측해로 쓰일 수가 있다. 비록 $N_t/2$개의 시구간 최적제어 문제의 해를 구하는데 시간은 더 걸렸지만 좋은 예측해의 선택으로 최적화 알고리즘이 N_t개의 시구간 최적제어 문제의 최적해를 빠르게 찾는다면 총 연산시간은 짧아질 수 있다. 이와 같은 방법은 연산시간 절감의 차원에서뿐만 아니라 최적화 알고리즘이 최적해에 수렴하지 못하고 중간에서 실패하는 경우를 방지하는데도 도움이 되며 이는 본 연구의 시뮬레이션을 통해서도 입증되었다.

III. 결과 및 고찰

본 연구에서는 병렬 최적제어 알고리즘을 사용하여 반도체 웨이퍼의 열처리 문제[8]의 최적해를 계산하였다. 웨이퍼 열처리 시스템은 원형 박판의 웨이퍼와 그 위에 놓여있는 3개의 반지름이 다른 원형 고리 모양의 열원들로 이루어 진다.

이 시스템은 좌우 대칭 형상을 가지므로 좌우중 한쪽 부분만을 고려할 수 있다. 공간좌표는 x로 나타내고, $x=0$은 웨이퍼의 중심을 $x=1$은 웨이퍼의 가장자리를 나타낸다. 웨이퍼의 온도를 $T(x,t)$로 나타낼 수 있다. 열원은 $s(x,t)$로 나타내며 세 개의 열원을 원의 중심에 가까운 것부터 $\bar{s}_1$, $\bar{s}_2$, $\bar{s}_3$로 나타낼 때 이들은 (40)의 관계식을 만족한다고 가정한다.

$$s(x,t)=\begin{cases}\bar{s}_1(t), & x\in[0,\delta_x]\\ \bar{s}_2(t), & x\in[0.5,0.5+\delta_x]\\ \bar{s}_3(t), & x\in[1-\delta_x,1]\end{cases} \tag{40}$$

열원은 일정한 범위의 값만을 가진다고 가정한다.

$$0\le s(x,t)\le s_{\max} \tag{41}$$

웨이퍼의 온도는 다음과 같은 포물선형 편미분방정식을 만족한다.

$$\frac{\partial T}{\partial t}=a(T)\frac{\partial^2 T}{\partial x^2}-k_1(T-T_a)+s(x,t) \tag{42}$$

$$a(T)=a_0(1+\beta T^{\gamma}) \tag{43}$$

윗 식에서 T_a는 주변온도를 나타내며 a와 k_1은 열전달에 관계된 상수이다. 이 중 a는 인위적으로 (43)과 같은 T의 비선형 함수로 만들었다. 위에서 β와 γ는 상수이다. T는 (44), (45)와 같은 경계조건을 만족하며, 시간에 대한 초기조건으로 (46)을 가정하였다.

$$\frac{\partial T(0,t)}{\partial x} = 0 \tag{44}$$

$$\frac{\partial T(1,t)}{\partial x} = -k_2(T(1,t) - T_a) \tag{45}$$

$$T(x,0) = 0 \tag{46}$$

윗 식에서 k_2는 열전달 관련 상수이다.

웨이퍼 열처리 문제의 목표는 웨이퍼의 온도가 [0, $t_{\max}$]동안 주어진 목표 온도변화 $T_g(t)$를 따라가도록 $\bar{s}_1(t)$, $\bar{s}_2(t)$, $\bar{s}_3(t)$를 찾는 것이다. 따라서 제어목표는 다음의 비용함수를 최소화하는 것과 같다.

$$\Phi = \int_0^{t_{\max}} \int_0^1 \{T(x,t) - T_g(t)\}^2 \, dx \, dt \tag{47}$$

위와 같이 정의된 웨이퍼 열처리 문제를 본 연구의 최적제어 알고리즘으로 풀기 위해서는 먼저 공간에 대해 이산화를 하여야 한다. 이를 위해 먼저 x의 구간 [0,1]을 N_x개의 균등한 세부구간으로 나눌 수 있다.

$$x_m = m\Delta x, \quad m = 0, \ldots, N_x \tag{48}$$

$$\Delta x = \frac{1}{N_x} \tag{49}$$

위와 같은 공간영역의 분해로 $s(x,t)$와 $T(x,t)$는 다음과 같이 이산화될 수 있다. 아래 식에서 $n_\delta = \Delta x/\delta_x$이다.

$$s_m(t) = \begin{cases} \bar{s}_1(t), m = 1, \ldots, n_\delta \\ \bar{s}_2(t), m = \frac{N_x}{2}, \ldots, \frac{N_x}{2} + n_\delta - 1 \\ \bar{s}_3(t), m = N_x - n_\delta, \ldots, N_x - 1 \end{cases} \tag{50}$$

$$T_m(t) = T(x_m, t), \quad m = 0, \ldots, N_x \tag{51}$$

(42)의 운동방정식을 x에 대해 유한차분법으로 근사화하면 (52)와 같은 미분방정식을 얻게 된다.

$$\frac{dT_m}{dt} = \frac{a(T_m)}{\Delta x^2}\{T_{m-1} + T_{m+1} - 2T_m\} -$$

$$k_1\{T_m - T_a\} + s_m, \quad m = 1, \ldots, N_x - 1 \tag{52}$$

(44)-(46)의 경계조건과 초기조건은 (53)-(55)와 같이 공간상에서 이산화되며 (41)의 구속조건은 (56)과 같이 이산화된다.

$$T_0(t) = T_1(t) \tag{53}$$

$$T_{N_x}(t) = \frac{1}{1 + k_2\Delta x}(T_{N_x-1}(t) + k_2\Delta x T_a) \tag{54}$$

$$T_m(0) = 0, \quad m = 0, \ldots, N_x \tag{55}$$

$$0 \le s_m(t) \le s_{\max}, \quad m = 1, \ldots, N_x - 1 \tag{56}$$

(47)의 비용함수는 다음과 같이 공간상에서 이산화될 수 있다.

$$\Phi = \int_0^{t_{\max}} \sum_{m=0}^{N_x} \rho_m \{T_m(t) - T_g(t)\}^2 \, dt \tag{57}$$

윗 식에서 ρ_m은 m에 따라 값이 결정되는 상수이다.

위에서 본 이산화는 편미분방정식을 상미분방정식으로 변환하기 위한 공간상의 이산화로 최적제어 알고리즘은 이와 같은 문제를 풀기 위해 시간에 대해 다시 한 번 이산화를 수행하게 된다.

본 연구에서는 위와 같은 웨이퍼 열처리 문제를 풀기 위해 $N_x = 10$으로 공간을 분할하였으며 $N_t = 16$으로 시구간을 분할하였다. 제어변수는 각 시구간에서 일정한 값을 갖도록 모델링하였다. 이같은 설정으로 총 변수의 수는 224개가 되며 각 시구간에서 적분해야 하는 미분방정식의 수는 135개가 된다.

본 연구에서는 Cray T3D 초병렬처리 컴퓨터를 사용하여 위의 문제를 풀었다. 최적해와 Lagrange multiplier의 초기 예측값을 모두 0으로 주었을 때 최적제어 알고리즘은 주반복 12회 부반복 216회만에 최적해를 찾았다.

그림 3은 종프로세서의 수를 달리하였을 때 병렬 최적화 알고리즘의 연산시간을 나타낸다. 그림은 전체 계산시간 중 메시지 교환을 위해 사용된 시간(PVM time)과 이를 제외한 나머지 시간(computation time)을 보여준다.

그림에서 종프로세서의 수가 2배씩 늘어날 때마다 순수 계산시간은 거의 2배씩 빨리지는 것을 볼 수 있다. 이것은 수치적분 연산이 전체 계산시간의 대부분을 차지한다는 것을 의미하는 것이다. 반면 프로세서의 수가 증가함에 따라 메시지 교환 횟수가 증가하여 메시지 교환 시간은 증가하는 것을 볼 수 있다. 순수 계산시간과 메시지 교환시간을 더한 전체 연산시간을 비교하면 16개의 프로세서로 2개의 프로세서 때보다 3.6배의 속도증가를 얻은 것을 알 수 있다. 그림 3은 2.4에서 소개된 효율적인 메시지 교환 방법을 채택하였을 때의 결과이다. 16개 프로세서를 사용하였을 때 이 메시지 교환 방법의 채택으로 전체 연산시간이 33% 절감되는 것을 볼 수 있었다.

그림 3에서 8개의 프로세서와 16개의 프로세서를 사용하였을 때 전체 연산시간 상에 큰 차이가 없음을 알 수 있다. 이는 16개의 프로세서를 사용한 경우 메시지 교환시간이 순수 계산시간보다 클 만큼 과도하기 때문이다. 병렬 알고리즘 연구분야에서는 이와 같은 점을 감안

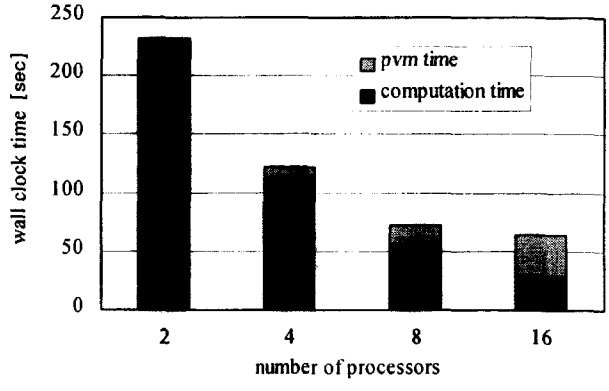
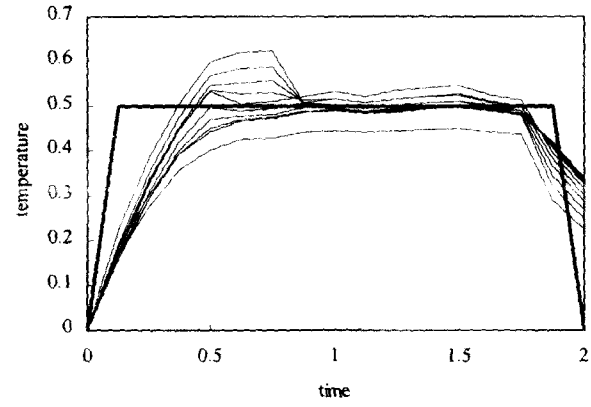
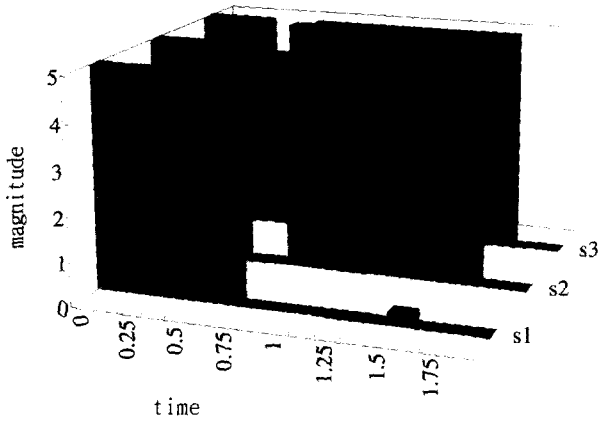


그림 3. 병렬 연산시간의 비교.
Fig. 3. Comparison of parallel computation times.



(a) state variables



(b) control variables

그림 4. 웨이퍼 열처리 문제의 최적해.
Fig. 4. Optimal solution of the wafer thermal processing problem.

하여 단순한 연산시간의 비교 이외에 병렬화 효율성이라는 개념을 함께 도입하는데 이는 병렬화에 따른 시간절감의 비를 프로세서의 증가비로 나눈 것이다. 그림 3에서 프로세서 2개의 경우를 기준으로 프로세서 4, 8, 16개의 경우를 보면 시간절감의 비는 각각 1.9배, 3.2배, 3.6배가 되고, 병렬화 효율성은 각각 95%, 81%, 45%가 된다.

그림 4는 웨이퍼 열처리 문제의 최적해를 나타낸다. 그림 4(a)에서 굵은 실선으로 표시된 부분은 원하는 웨이퍼의 목표온도 $T_g(t)$를 나타내며 가는 실선으로 표시된 부분은 웨이퍼의 온도를 나타낸다. $t=0.5$일 때를 기준으로 가장 위의 선이 웨이퍼 중심의 온도 $T_1(t)$ $(=T_0(t))$이고 가장 아래의 선이 웨이퍼의 가장자리 온도 $T_{10}(t)$이며 그 사이의 선들은 차례로 $T_2(t)$, ..., $T_9(t)$를 나타낸다.

그림 4(a)에서 시구간의 처음 부분을 보면 웨이퍼의 모든 점에서의 온도가 원하는 온도 $T_g(t)$보다 작음을 알 수 있다. 이는 (52)의 시스템이 주어진 열원의 최대값으로도 $t \in [0, 0.125]$ 구간에서의 $T_g(t)$의 빠른 변화를 따라갈 수 없기 때문이다. 그림 4(b)를 보면 처음 얼마동안 세 열원 모두가 최대의 출력을 내는 것을 볼 수 있다. 이와 비슷한 현상은 시구간의 마지막 부분에서도 일어난다. 이번에는 앞 부분과는 반대로 $T_g(t)$가 0.5에서 0으로 빠르게 감소하나 주어진 웨이퍼 시스템의 동적 특성으로는 이와 같은 급격한 감소를 따라갈 수가 없는 것을 볼 수 있다.

그림 4(a)를 보면 시구간의 마지막 부분에서 $T_g(t)$의 감소가 시작되기 전부터 웨이퍼의 온도가 감소하기 시작한다. 이와 같은 현상은 최적화 알고리즘이 주어진 웨이퍼 시스템의 동적특성으로는 앞으로 닥칠 $T_g(t)$의 급격한 감소를 추적할 수 없다는 것을 알기 때문에 미리부터 이에 대처한 것이다. 그림 4(a)를 보면 웨이퍼의 가장자리로 갈수록 온도가 낮은 것을 볼 수 있는데 이는 그 점에서 (54)와 같은 주위로의 열손실이 있기 때문이다. 이에 따라 열원도 가장자리에 가까운 것일수록 많은 출력을 내는 것을 그림 4(b)에서 볼 수 있다.

본 연구에서는 위에서 정의된 웨이퍼 열처리 문제를 대상으로 2.4절에서 제안된 Jacobian 재사용법을 시도하였다. 이를 위해 웨이퍼 열처리 문제를 공간과 시간에 대해 모두 10개의 구간으로 이산화하고 이를 단일 CPU에서 계산하였다. 매 주반복점에서 Jacobian을 계산한 경우 최적해를 찾기까지 주반복 10회, 부반복 107회가 수행되었고 Jacobian을 재사용한 경우 주반복 22회, 부반복 117회가 수행되었다. 그러나 계산시간에 있어서는 후자가 전자에 비해 약 30%의 시간절감을 가져다 주었다. 이는 매우 고무적인 결과로 Jacobian의 재사용법은 특히 대규모 최적화 문제의 계산시간을 줄이는데 도움을 줄 수 있다.

Ⅳ. 결론

본 연구에서는 대규모 동적 시스템의 최적제어 문제를 빠르게 풀기 위한 병렬 최적제어 알고리즘을 개발하였다. 이를 위해 본 연구에서는 먼저 연속시간 상에서 정의된 최적제어 문제를 이산화하고 이산화된 최적제어 문제에 비선형 프로그래밍 기법을 적용하였다. 최적화 알고리즘으로는 Sequential Quadratic Programming(SQP) 알고리즘을 사용하였으며 최적화 과정에서 필요한 수치적분 연산을 위해서는 DASPKSO를 사용하였다.

본 연구에서는 대규모 최적제어 문제의 해를 빠르게 찾기 위해 전체 수치연산의 가장 많은 부분을 차지하는 수치적분 연산부를 영역분해 방식으로 병렬화하였다. 메시지 교환에 드는 시간을 줄이기 위해 효율적인 메시지 교환을 시도하였다.

본 연구에서 개발된 병렬 최적제어 알고리즘이 가지는 특징을 정리하면 다음과 같다. (1)-(4)와 같은 최적제어 문제에 있어서 비선형의 함수, 등식형 구속조건, 부등식형의 구속조건을 모두 다룰 수 있다. 특히 구속조건 중 상태변수에 의존하는 함수의 경우 최적제어 알고리즘에 어려움을 주기 때문에 여러 연구에서 이 경우를 제외하는 것을 볼 수 있는데[4] 본 연구의 영역분해법에서는

적절한 변수 이산화를 통해 이러한 구속조건을 가지는 최적제어 문제를 풀 수 있도록 하였다. 본 연구의 알고리즘은 그림 1에서 보는 바와 같이 임의의 상태변수의 시간함수를 예측해로 사용할 수 있는데 이러한 성질은 비선형성이 심한 시스템을 다루는 경우 도움이 된다. 또한 본 연구에서는 추가적인 시간절감을 위한 몇 가지 실용적인 방법이 연구되었는데 효율적인 메시지 교환, Jacobian의 재사용, 이산구간의 해상도를 점진적으로 늘려가며 예측해를 찾는 법 등이 그것이다. 특히 이산구간의 해상도를 점진적으로 늘리는 방법은 알고리즘이 기존의 예측해로부터 출발하여 최적해에 수렴하지 못하는 경우, 보다 나은 예측해를 찾아 알고리즘의 수렴성을 높이는데 유용하게 사용될 수 있다.

본 연구에서는 병렬 최적제어 알고리즘을 사용하여 반도체 웨이퍼의 열처리 문제의 최적해를 계산하였다. Cray T3D 초병렬처리 컴퓨터 상에서 프로세서 16개를 사용했을 때 프로세서 2개를 사용했을 때보다 45%의 병렬효율성에 해당하는 3.6배의 속도증가를 얻었다. 또한 효율적인 메시지 교환을 사용하여 연산시간을 추가로 33% 절감하였다.

본 연구에서는 병렬화와 더불어 최적화 연산시간을 줄이기 위한 실용적인 방법에 대해서도 고찰하였으며 시뮬레이션을 통해 이들 방법의 타당성을 검증하였다.

참고문헌

[1] R. E. Larson and E. Tse, "Parallel processing algorithms for the optimal control of nonlinear dynamic systems," *IEEE Trans. Computers*, vol. C-22, no. 8, pp. 777-786, 1973.

[2] R. Travassos and H. Kaufman, "Parallel algorithms for solving nonlinear two-point boundary-value problems which arise in optimal control," *Journal of Optimization Theory and Applications*, vol. 30, pp. 53-71, 1980.

[3] J. T. Betts and W. P. Huffman, "Trajectory optimization on a parallel processor," *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 2, pp. 431-439, 1991.

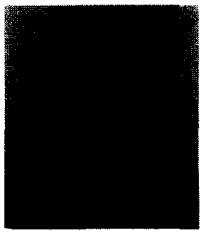
[4] S. J. Wright, "Partitioned dynamic programming for optimal control," *SIAM Journal on Optimization*, vol. 1, pp. 620-642, 1991.

[5] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, Academic Press, New York, 1981.

[6] Maly, T. and L. R. Petzold, "Numerical methods and software for sensitivity analysis of differential-algebraic systems," *Applied Numerical Mathematics*, vol. 20, pp. 57-79, 1996.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine*, The MIT Press, 1994.

[8] C. D. Schaper, Y. M. Cho and T. Kailath, "Low-order modeling and dynamic characterization of rapid thermal processing," *Applied Physics*, A54, pp. 317-326, 1992.



박 기 홍

1986년 서울대 기계설계학과 졸업(학사). 1990년 Cornell University 기계공학과 졸업(석사). 1994년 Cornell University 기계공학과 졸업(박사). 1995년-현재 국민대학교 기계자동차공학부 조교수. 관심분야는 병렬처리, 최적제어, 반능동형 현가장치 제어, 차량 동력학 제어.