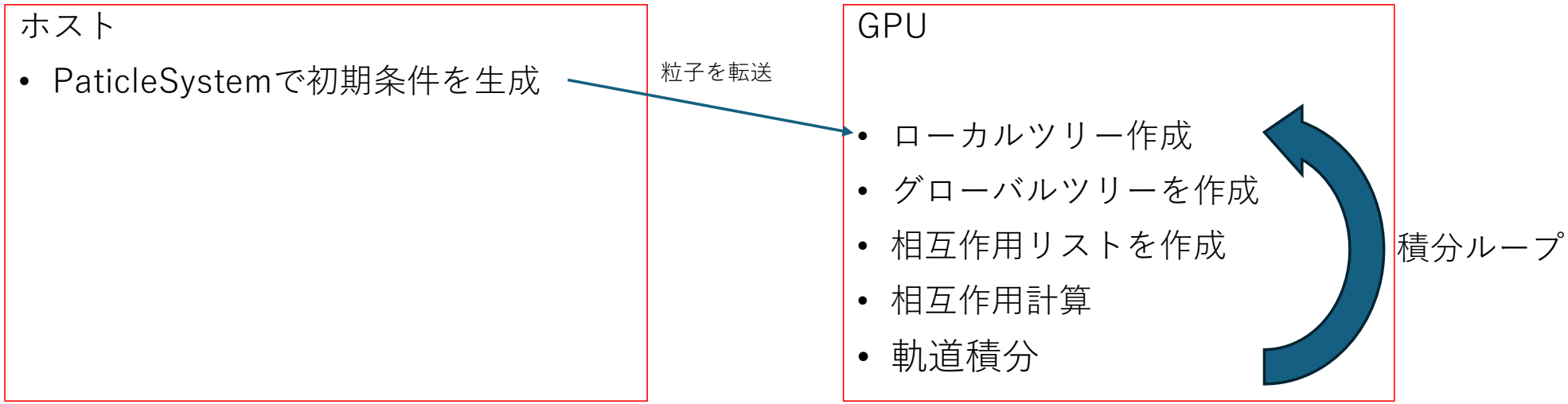
CUDA版FDPS

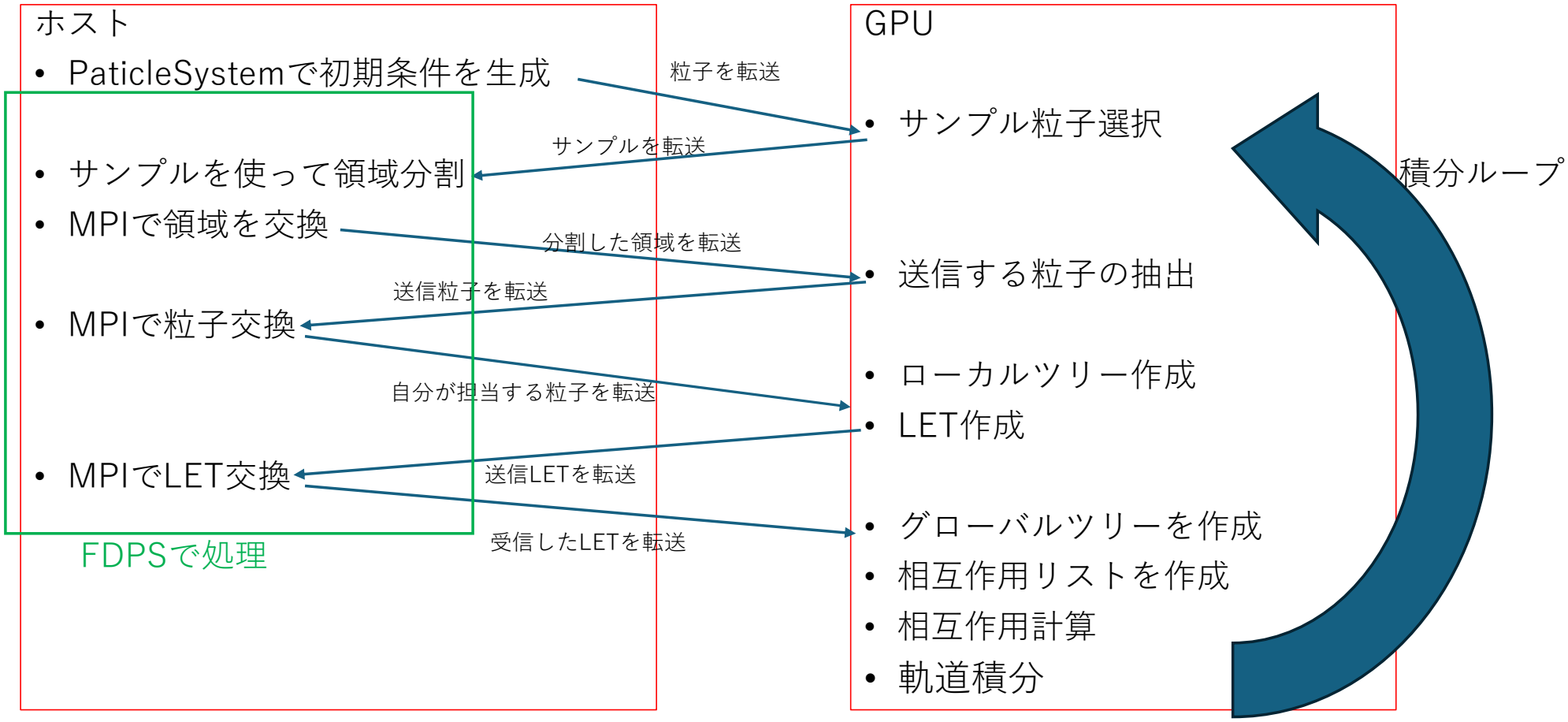
- FDPS Ver.9からTree/相互作用リスト構築など全てNvidiaのGPUで計算できるようになる。
- FP, EP, SPの配列がGPUのデバイスメモリに載る.
  - GPUでtree作成、相互作用リスト作成可能

Single GPUの計算の流れ



GPU内で軌道積分のループが完結する

multi GPUの計算の流れ



プログラムの主な変更点

- FP, EP, SPのメンバ関数にPS_HOST_DEVICEをつける
  - PS_HOST_DEVICEをつける事でGPU上で動作するようになる.
- 名前空間をPSからPS_CUDAに変更する.
  - PS_CUDA::decomposeDomainAll
  - PS_CUDA::exchangeParticle
  - PS_CUDA::calcForceAllAndWriteBack
- 相互作用カーネルはVer.9用に変更
- FPの軌道積分
  - GPUで行う：ParticleSystem::getDevicePointer()によりデバイスポインタを取得して軌道積分
  - ホストで行う：ParticleSystem::d2h()でGPUのFPの情報をホスト側にコピーする。軌道積分後ParticleSystem::h2d()でホストのFPの情報をGPUに転送

コンパイル方法

- 以下のコンパイルオプションをつける
  - -DPARTICLE_SIMULATOR_ENABLE_CUDA
    - CUDAライブラリの有効か
  - -x cu
    - 入力ファイルを.cuファイルとして解釈
  - --extended-lambda
    - ラムダ関数をデバイスコードで使える
  - –ccbin $(CXX)
    - C++部分のコンパイルに使うコンパイラを指定

コンパイル例

nvcc -DPARTICLE_SIMULATOR_ENABLE_CUDA –ccbin mpicxx –x cu --extended-lambda nbody.cpp

重力N体の例(粒子クラスの定義)

```
class FP{
  PS::F64vec pos;
  PS::F64vec vel;
  PS::F64vec getPos();
  PS::F64vec getCharge();
  PS::F64vec clear();
};
```

```
class FP{
  PS::F64vec pos;
  PS::F64vec vel;
  PS_HOST_DEVICE PS::F64vec getPos();
  PS_HOST_DEVICE PS::F64vec getCharge();
  PS_HOST_DEVICE PS::F64vec clear();
};
```

重力N体の例(相互作用関数)

```
struct GravityOnCPU {
  static inline PS::F32 eps2 = 1.0f / (64.0f * 64.0f);
  template <class Tforce, class Tepi, class Tepj>
  void operator()(const Tepi * epi, const PS::S32 n_epi,
                  const Tepj * epj, const PS::S32 n_epj,
                  Tforce * force) {
  for(PS::S32 i = 0; i < n_ip; i++){
    PS::F64vec xi = ep_i[i].getPos();
    PS::F64vec ai = 0.0;
    PS::F64 poti = 0.0;
    for(PS::S32 j = 0; j < n_jp; j++){
      PS::F64vec rij    = xi - ep_j[j].getPos();
      PS::F64    r3_inv = rij * rij + eps2;
      PS::F64    r_inv  = 1.0/sqrt(r3_inv);
      r3_inv  = r_inv * r_inv;
      r_inv   *= ep_j[j].getCharge();
      r3_inv *= r_inv;
      ai       -= r3_inv * rij;
      poti   -= r_inv;
    }
    force[i].acc += ai;
    force[i].pot += poti;
  }
}
};
```

```
struct GravityOnGPU {
  static inline PS::F32 eps2 = 1.0f / (64.0f * 64.0f);
  template <class Tforce, class Tepi, class Tepj, class Tspj>
  void operator()(Tforce* force, const PS::S32* id_walk,
                  const PS::S32 n_epi, const Tepi* epi,
                  const Tepj* epj, const PS::S32* adr_epj,
                  const PS::S32* n_epj, const PS::S32* n_disp_epj,
                  const Tspj* spj, const PS::S32* adr_spj,
                  const  PS::S32* n_spj, const PS::S32* n_disp_spj) {
    if (n_epi_tot <= 0) return;
    constexpr PS::S32 n_thr = 256;
    const PS::S32 n_blk = (n_epi + n_thr– 1)/n_thr;
    KernelGravity_1PtclPerThread<<<n_blk, n_thr>>>
    (force, id_walk, epi, n_epi, epj, n_disp_epj, adr_epj,
    spj, n_disp_spj, adr_spj, eps2);
  }
};
```

重力N体の例(メイン関数)

```
int main(){
  PS::ParticleSystem sys;
  // 初期条件
  sys[0].mass = 1/N;
  // …


  PS::DomainInfo dinfo;
  dinfo.decomposeDomainAll();
  sys.exchangeParticle();
  PS::TreeForForce tree;
  tree.calcForceAllAndWriteBack();
  update(sys); // 軌道積分
}
```

```
int main(){
  PS_CUDA::ParticleSystem sys;
  // 初期条件
  sys[0].mass = 1/N; // sys[]はホストメモリを指す
  //…
  sys.h2d(); // ホストからGPUへ転送

  PS_CUDA::DomainInfo dinfo;
  dinfo.decomposeDomainAll();
  sys.exchangeParticle();
  PS_CUDA::TreeForForce tree;
  tree.calcForceAllAndWriteBack();
  update_dev(sys); //軌道積分.
}
```

重力N体の例(メイン関数)

```
int main(){
  PS::ParticleSystem sys;
  // 初期条件
  sys[0].mass = 1/N;
  // …


  PS::DomainInfo dinfo;
  dinfo.decomposeDomainAll();
  sys.exchangeParticle();
  PS::TreeForForce tree;
  tree.calcForceAllAndWriteBack();
  update(sys); // 軌道積分
}
```

```
int main(){
  PS_CUDA::ParticleSystem sys;
  // 初期条件
  sys[0].mass = 1/N; // sys[]はホストメモリを指す
  //…
  sys.h2d(); // ホストからGPUへ転送

  PS_CUDA::DomainInfo dinfo;
  dinfo.decomposeDomainAll();
  sys.exchangeParticle();
  PS_CUDA::TreeForForce tree;
  tree.calcForceAllAndWriteBack();
  update_dev(sys); //軌道積分.
}
```

重力N体シミュレーション(軌道積分)

```
void update(sys){
  for(int i=0; i<n; i++){
    sys[i].vel += sys[i].acc*dt;
    sys[i].pos += sys[i].vel*dt;
  }
}
```

- ホストで積分

```
void update_dev(sys){
  sys.d2h();
  for(int i=0; i<n; i++){
    sys[i].vel += sys[i].acc*dt;
    sys[i].pos += sys[i].vel*dt;
  }
  sys.h2d();
}
```

- GPUで積分

```
void update(sys){
  FP * pdev = sys.getDevicePointer();
  PS::S32 nthr = 256;
  PS::S32 nblk = (n+nthr-1) / nthr;
  update_kernel<<<nblk, nthr>>>(pdev);
}
__global__ update_kernel(FP* pdev){
  PS::S32 i= blockIdx.x * blockDim.x + threadIdx.x;
  pdev[i].vel += pdev[i].acc*dt;
  pdev[i].pos += pdev[i].vel*dt;
}
```