Universidad Nacional de Córdoba

FaMAF | GPGPU Computing Group

Integrating Post-Newtonian Equations

Matías Bellone et al.

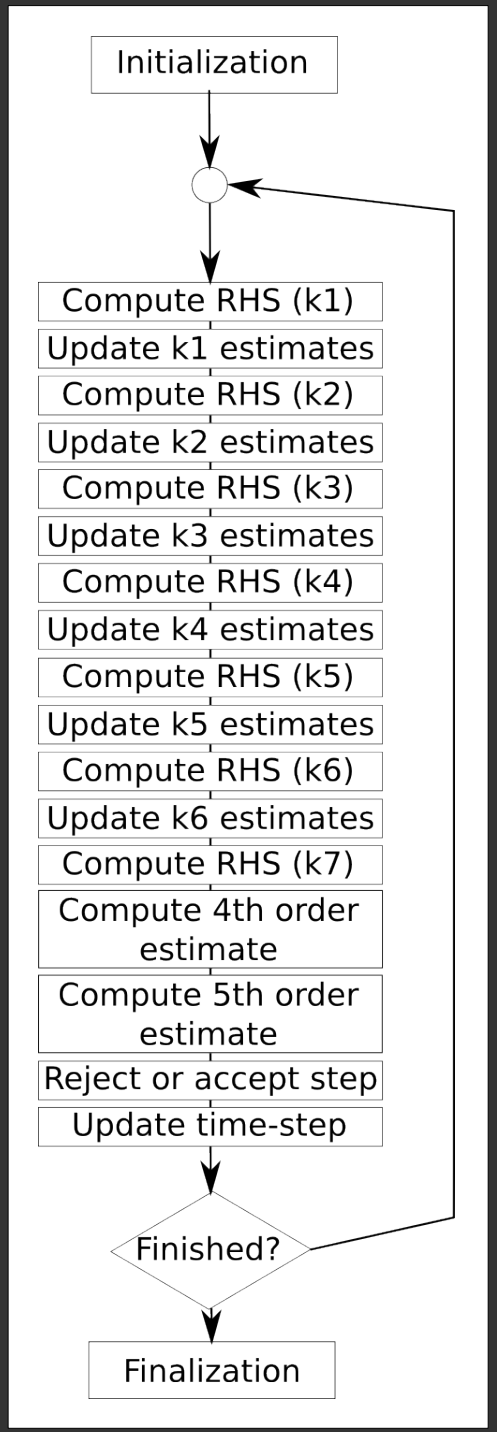
¿Qué?

$$
\begin{aligned}
\dot{\omega} &= \omega^2 \frac{96}{5}\eta (M\omega)^{5/3} \Bigg\{ 1 - \frac{743+924\eta}{336}(M\omega)^{2/3} \\
&\quad - \left(\frac{1}{12} \sum_{i=1,2} \left(\chi_i \hat{\mathbf{L}}_n \cdot \hat{\mathbf{S}}_i (\frac{113 m_i^2}{M^2} + 75\eta) \right) - 4\pi \right) M\omega \\
&\quad + \left(\frac{34103}{18144} + \frac{13661}{2016}\eta + \frac{59}{18}\eta^2 \right) (M\omega)^{4/3} \\
&\quad - \frac{1}{48}\eta\chi_1\chi_2 \left(247(\hat{\mathbf{S}}_1 \cdot \hat{\mathbf{S}}_2) - 721(\hat{\mathbf{L}}_n \cdot \hat{\mathbf{S}}_1)(\hat{\mathbf{L}}_n \cdot \hat{\mathbf{S}}_2) \right) (M\omega)^{4/3} \\
&\quad - \frac{1}{672}(4159 + 15876\eta)\pi (M\omega)^{5/3} \\
&\quad + \Bigg(\left(\frac{16447322263}{139708800} - \frac{1712}{105}\gamma_E + \frac{16}{3}\pi^2 \right) + \left(-\frac{273811877}{1088640} + \frac{451}{48}\pi^2 - \frac{88}{3}\hat{\theta}\eta \right) \eta \\
&\quad + \frac{541}{896}\eta^2 - \frac{5605}{2592}\eta^3 - \frac{856}{105} log(16(M\omega)^{2/3}) \Bigg) (M\omega)^2 + (-\frac{4415}{4032} + \frac{358675}{6048}\eta + \frac{91495}{1512}\eta^2)\pi (M\omega)^{7/3} \Bigg\} \\
\dot{\mathbf{S}}_i &= \mathbf{\Omega}_i \times \mathbf{S}_i \\
\dot{\hat{\mathbf{L}}}_n &= -\frac{(M\omega)^{1/3}}{\eta M^2}\frac{d\mathbf{S}}{dt}
\end{aligned}
$$

and $\mathbf{\Omega}_1$ given by: $\mathbf{\Omega}_1 = \frac{(M\omega)^2}{2M}\left(\eta (M\omega)^{-1/3}(4 + 3\frac{m_2}{m_1})\hat{\mathbf{L}}_n + 1/M^2(\mathbf{S}_2 - 3(\mathbf{S}_2 \cdot \hat{\mathbf{L}}_n)\hat{\mathbf{L}}_n) \right)$

Ahhh...

- 2 agujeros negros
- Aproximación post-newtoniana: expansión a serie de Relatividad General de Einstein
- Resultado: Sistema de ODEs acopladas a resolver
- ODE: Ecuaciones Diferenciales Ordinarias
- Algoritmo Dormand-Price
  - Standard
  - Integrador runge-kutta de 4to orden
  - Paso temporal variable



Particularidades

- Es **trivialmente paralelizable**
- No existe comunicación
- No hay sincronización
- Ejecutamos muchos ODEs en paralelo
- ¿Cuántos? Todos los que podamos

Secuencial

```
while (1) {
        nstp++;
        assert(h >= 0);
        if (time >= t2 || h < hmin || ierr != 0) break;
        if (time+h > t2) h = t2-time;

        // compute slopes k1...k7
        ierr += f_rhs(y, k1, OMEGA_FINAL);
        for (i = 0; i < N; i++)
                ytmp[i] = y[i]+h*a21*k1[i];
        ierr += f_rhs(ytmp, k2, OMEGA_FINAL);
        for (i = 0; i < N; i++)
                ytmp[i] = y[i]+h*(a31*k1[i]+a32*k2[i]);
        ierr += f_rhs(ytmp, k3, OMEGA_FINAL);
        for (i = 0; i < N; i++)
                ytmp[i] = y[i]+h*(a41*k1[i]+a42*k2[i]+a43*k3[i]);
        ierr += f_rhs(ytmp, k4, OMEGA_FINAL);
        for (i = 0; i < N; i++)
                ytmp[i] = y[i]+h*(a51*k1[i]+a52*k2[i]+a53*k3[i]+a54*k4[i]);
        ierr += f_rhs(ytmp, k5, OMEGA_FINAL);
        for (i = 0; i < N; i++)
                ytmp[i] = y[i]+
                        h*(a61*k1[i]+a62*k2[i]+a63*k3[i]+a64*k4[i]+a65*k5[i]);
        ierr += f_rhs(ytmp, k6, OMEGA_FINAL);
        for (i = 0; i < N; i++)
                ytmp[i] = y[i]+
                        h*(a71*k1[i]+ +a73*k3[i]+a74*k4[i]+a75*k5[i]+a76*k6[i]);
        ierr += f_rhs(ytmp, k7, OMEGA_FINAL); // note that k2 is used for k7
```

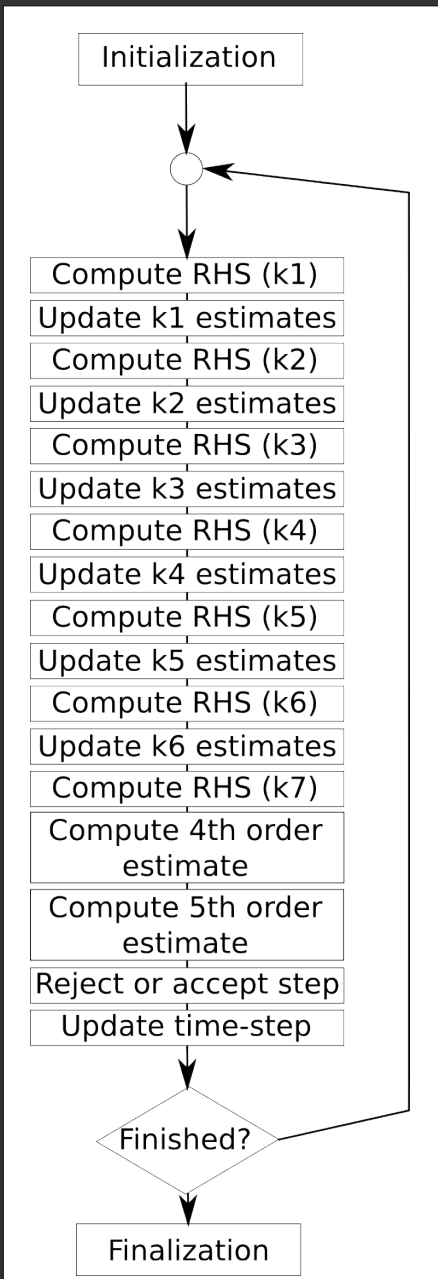
```
// 4th order estimate
for (i = 0; i < N; i++)
        y4[i] = y[i]+h*(b4_1*k1[i]+b4_3*k3[i]+b4_4*k4[i]+
                        b4_5*k5[i]+b4_6*k6[i]+b4_7*k7[i]);
// 5th order estimate
for (i = 0; i < N; i++)
        y5[i] = y[i]+h*(b5_1*k1[i]+b5_3*k3[i]+b5_4*k4[i]+
                        b5_5*k5[i]+b5_6*k6[i]);

if (ierr == 0) { // all fine on the RHS
        // compare truncation error und acceptable error
        delta = -1e10;
        yinf = -1e10;
        for (i = 0; i < N; i++) {
                dif = fabs(y5[i]-y4[i]);
                if (dif > delta) delta = dif;
                ya = fabs(y[i]);
                if (ya > yinf) yinf = ya;
        }
        tau = fmax(yinf, 1)*tol;

        if (delta <= tau) { // acceptable error
                time = time+h; // use 5th order estimate for y
                for (i = 0; i < N; i++) y[i] = y5[i];
        }

        if (delta == 0) delta = 1e-16;
        if (h != 0) {
                // single precision
                //h=fmin(hmax,0.8*h*powf(tau/delta,power));

                // double precision
                h = fmin(hmax, 0.8*h*pow(tau/delta, power));
        }
} else { // something wrong with the RHS
        ierr = 0; // reset
        if (fabs(y[0] - OMEGA_FINAL) < OMEPS) {
                h = 0; // we reached the final value of omega
        } else {
                h /= 2; // if omega not reached, tried half step
        }
}
```

Paralelo

```
while (1) {
  nstp++;

  update_time_step<<<nBlocksParts,nThreadsParts>>>(hmin,time,h,t2,istop,Nvars,Nparts);

  const int done=check_all_done(h,tmp_h,Nparts);
  if (done>0) break;

  // compute slopes k1...k7
  const int N=Nvars*Nparts;
  f_rhs<<<nBlocksParts,nThreadsParts>>>(y,k1,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k1<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k2,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k2<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k3,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k3<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k4,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k4<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,k4,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k5,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k5<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,k4,k5,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k6,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k6<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,k4,k5,k6,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k2,masses,chi1s,chi2s,Nvars,Nparts,istop);
  float *k7=k2; // recycle k2 memory

  // 4th order estimate
  estimate_y4<<<nBlocksVars,nThreadsVars>>>(y4,y,h,k1,k3,k4,k5,k6,k7,N,Nparts);
  // 5th order estimate
  estimate_y5<<<nBlocksVars,nThreadsVars>>>(y5,y,h,k1,k3,k4,k5,k6,k7,N,Nparts);

  // update time, time-step h xxx. size
  update_time_h<<<nBlocksParts,nThreadsParts>>>(hmax,tol,time,h,y4,y5,y,Nvars,Nparts,istop);
}
```

Paralelo

```
while (1) {
  nstp++;

  update_time_step<<<nBlocksParts,nThreadsParts>>>(hmin,time,h,t2,istop,Nvars,Nparts);

  const int done=check_all_done(h,tmp_h,Nparts);
  if (done>0) break;

  // compute slopes k1...k7
  const int N=Nvars*Nparts;
  f_rhs<<<nBlocksParts,nThreadsParts>>>(y,k1,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k1<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k2,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k2<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k3,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k3<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k4,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k4<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,k4,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k5,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k5<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,k4,k5,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k6,masses,chi1s,chi2s,Nvars,Nparts,istop);
  y_k6<<<nBlocksVars,nThreadsVars>>>(ytmp,y,h,k1,k2,k3,k4,k5,k6,N,Nparts);
  f_rhs<<<nBlocksParts,nThreadsParts>>>(ytmp,k2,masses,chi1s,chi2s,Nvars,Nparts,istop);
  float *k7=k2; // recycle k2 memory

  // 4th order estimate
  estimate_y4<<<nBlocksVars,nThreadsVars>>>(y4,y,h,k1,k3,k4,k5,k6,k7,N,Nparts);
  // 5th order estimate
  estimate_y5<<<nBlocksVars,nThreadsVars>>>(y5,y,h,k1,k3,k4,k5,k6,k7,N,Nparts);

  // update time, time-step h xxx. size
  update_time_h<<<nBlocksParts,nThreadsParts>>>(hmax,tol,time,h,y4,y5,y,Nvars,Nparts,istop);
}
```

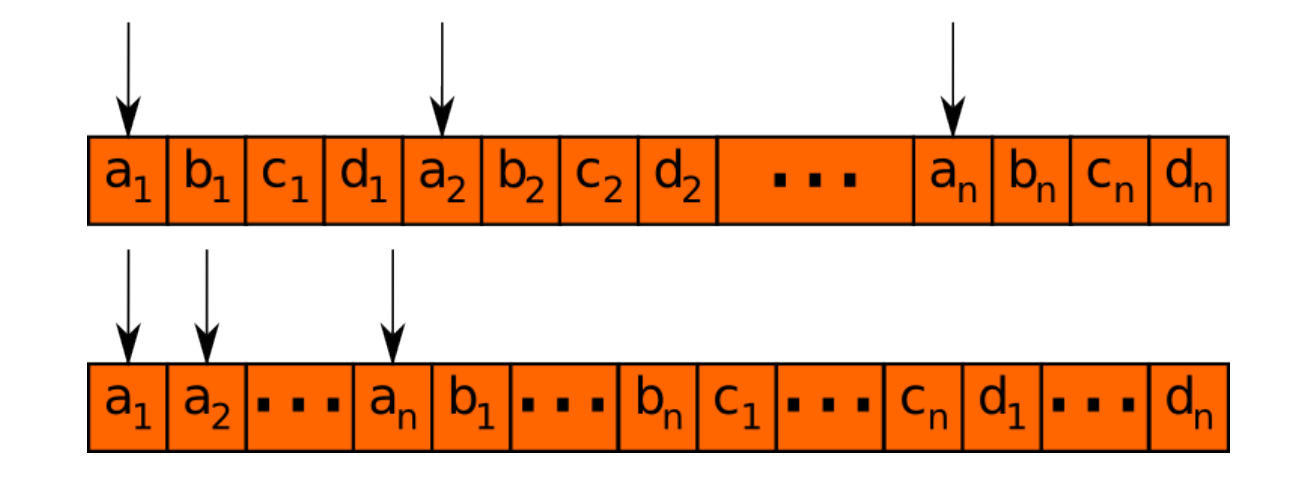


Más paralelo todavía

- El algoritmo permite más paralelismo
- Cada estimado se hace por variable
  - Son independientes entre sí
- Paralelicemos esos kernels por variable
  - Volvemos al paradigma de 1 thread por dato
  - Eliminamos un bucle y su variable interna
  - El código es un poquito más complejo

Recordando el Hardware

- Los accesos a memoria no son los ideas
- Revisar accesos a una variable de un sistema
- El RHS por ejemplo
- Cambiar la forma en la que está ordenado
- Sólo cambia el mapeo de thread a dato, nada más



Performance